Trusted embedded System Operating System

(TeSOS)

Study and Design

System Security Group
Ruhr-University Bochum

Sirrix AG security technologies

Bundesamt für Sicherheit in der Informationstechnik

Revision 506

Trusted embedded System Operating System (TeSOS)

Study and Design

Ahmad-Reza Sadeghi, Steffen Schulz, Alexandra Dmitrienko
Chair for System Security, Ruhr-University Bochum

Christian Stüble, Dennis Gessner
Sirrix AG security technology

Markus Ullmann
Bundesamt für Sicherheit in der Informationstechnik

Contents

1 Introduction

A Wireless Sensor Network (WSN) is an infrastructure of physically separated sensor nodes that communicate through wireless network technology. Sensor nodes in this context are typically low cost highly integrated embedded devices with limited power carrying out sensing and data processing tasks. They typically consist of a power source, a processing unit, a sensing unit and a radio transceiver for node-to-node communication. Restrictions on deployment capabilities and energy available in the target environment impose strong limits on hardware features of sensor nodes and power consumption. The resulting limits on available computational performance and communication abilities are the main characteristics of WSNs.

Due to advances in wireless communication and in the miniaturization of electronic components, sensor network technology has grown rapidly during the last few years. The development of large scale sensor networks offers economically viable monitoring solutions for a wide range of applications including home, industry and environment control.

WSNs are increasingly considered for critical applications like surveillance of critical infrastructures, control of medical equipment or fast provisioning of reliable communication networks. In such areas, WSNs must be resistant against accidents and human error but also against targeted attacks. Hence WSNs must also feature security mechanisms such as device authentication, secure routing and secure bootstrapping. Reaching strong security on the one hand and the design and implementation of lightweight security mechanisms suitable for resource constrained devices on the other hand is a challenging task, and still subject of ongoing research.

Generally wireless networks are prone to security attacks since the nature of wireless communication easily allows eavesdropping and alteration of messages. Moreover, deployment of WSNs in a field where sensor nodes have to operate unattended assume that an adversary has physical access to sensor nodes and is able to launch physical attacks. Although different schemes have been proposed to encounter threats to confidentiality and integrity of data transmitting over the network, vulnerability of sensor nodes to physical compromise has not been well investigated yet.

1.1 Objectives and Outline

The goals of this project is to study the state-of-the-art of WSNs and to propose an integrated solution for a Trusted Embedded Secure Operating System (TeSOS) consisting of the architecture design for sensor nodes and lightweight security concept for a WSN.

We call the project TeSOS throughout this document. TeSOS consists of the following parts:

(1) WSN design overview.

(2) Security objectives and requirement analysis.

(3) Investigation of related work on WSN security, hardware architectures and open-source operating systems.

(4) Analysis of related work in order to identify if existing literature covers all required by TeSOS aspects.

(5) Developing of new protocols/mechanisms/paradigms in case if existing literature does not provide all required solutions to satisfy TeSOS security requirements.

(6) Developing an integrated concept for secure WSNs based on the preceding analysis.

This document is structured as following: Chapter 2 gives an overview on the common WSN design approaches and deployment strategies. Chapter 3 defines security objectives and analyses security requirements specific for this project. Chapter 4 provides the study of the related work on WSN security aspects. Chapter 5 concerns current microprocessor-architectures in order to clarify their applicability for secure WSNs. Chapter 6 analyzes the related work presented in Chapter 4 and concludes if existing literature covers all required by TeSOS security aspects. When required, new protocols/mechanisms/paradigms will be developed and introduced in this chapter. Chapter 7 provides comparative study of the nowadays operating systems for embedded systems. Chapter 8 integrates preceding results into a single integrated solution for secure WSN.

1.2 Sample Application Scenarios

Possible scenarios for TeSOS WSNs are listed below:

1. Monitoring of public places, temporary or permanent. Temporary monitoring is needed during high risk events, e.g., demonstrations, open air concerts. Permanent monitoring is required in crowded places such as airports, train stations and stadiums.

2. Border control, i.e., anti-intrusion perimeter systems to provide alerting and intrusion blocking facilities for important, vital, or restricted regions an areas (e.g., shipping detection and surveillance at the European coastline).

3. Situational monitoring systems, which enable the ability to identify, process and comprehend the critical information about an incident (e.g., a system to detect certain bioterrorism agents in air or water).

All listed applications focus on detecting crucial events, location sensing and object tracking. Some of these applications require permanent deployment of WSNs (e.g., border control), the others should be deployed on-demand (e.g., monitoring of high-risk events). Permanently deployed WSNs may be statical, while on-demand deployed networks should be highly scalable, extensible and self-organizing. Although functional requirements to these applications can differ, they have similar requirements on enhanced security and high network reliability.

1.3 Milestones

Milestone	Module	Work Package
1 - October 2009	Kick-Off Workshop	–
2 - January 2010	M1 - Assumptions and Goals	–
3 - March 2010	M2 - Related Work	–
	M3 - Hardware Evaluation	WP1
4 - November 2010	M4 - TeSOS Analysis	WP2
	M5 - OS Evaluation	WP3
5 - January 2011	M6 - Integration	WP4

2 General WSN Design

WSNs can be used in many different scenarios. This results in several different design approaches, deployment strategies and operational requirements. Here we describe possible network structure, architecture and topology, explain typical life cycle of WSNs and provide a definition for security objectives that are fundamental for WSNs.

2.1 WSN Architectures

WSNs can be distinguished according to their architecture as clustered (also named as hierarchical [cY05]) and non-clustered (also referred as distributed [cY05], decentralized [SWAG07] and flat [dAFN$^+$08]). Both, clustered and non-clustered WSNs, usually have a single central node to collect information from sensor nodes. This central node is often referred as Base Station (BS) and typically serves as a gateway to another network, a powerful data processing or storage center, or an access point for a human interface.

Figure 2.1 illustrates information flow in networks of both categories. A clustered WSN [YYA02] consists of clusters where special devices, so-called Cluster Heads (CHs), are typically used as fusion points for aggregation of data within clusters. They carry out in-network data processing in order to reduce the amount of data that is actually transmitted to the BS. Clustering can be purely static, when once elected, cluster heads serve for the entire lifetime of the network, or dynamic, when role of CH is periodically rotated randomly among all nodes, as proposed in LEACH [HCB00]. Dynamic clustering is applied to ensure that energy resources of all nodes will be exhausted at about the same time, so minimum efforts are required to replace empty batteries with new ones.

In contrast to clustered WSNs, non-clustered networks feature device nodes of equal functionality.

As clustered WSNs provide more powerful nodes, called "Base stations", in most cases it is not necessary for the sensor nodes to provide their own in-network data processing, in spite of it would be possible for them to simple process their data, even if they are not that high-performance, compared with the base stations.

Such approach may result in performance degradation when applied to large scale WSN due to overhead of sensor nodes located near to BS. The non-clustered architecture is typically used due to the limited physical access to the target area. The device nodes are scattered randomly in the target area so that the location of individual nodes cannot be determined prior to deployment. In other cases, if the physical absolute or relative location of device nodes is known prior to deployment, this knowledge can be used to optimize key management and communication patterns, often resulting in a clustered WSN architecture.

2.2 WSN Structures

Beside architecture differences, WSNs can differ with regards to capabilities of sensor nodes [MR04]. Networks consisting of identical sensor nodes in terms of battery energy and hardware complexity introduce type of networks with *homogeneous* structure. Networks, where resource-rich and

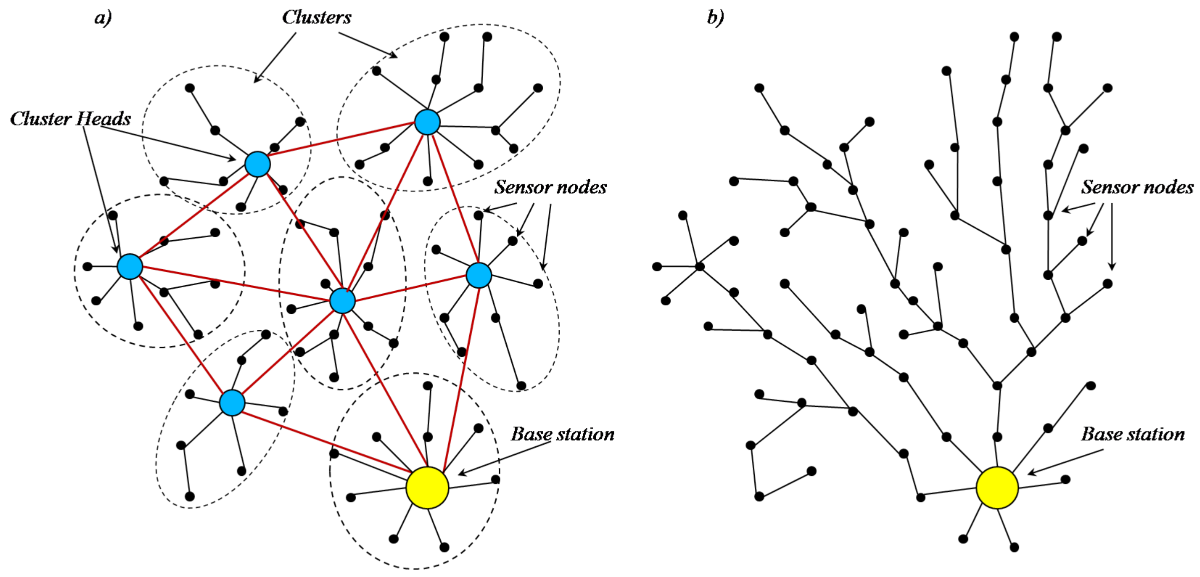


Figure 2.1: Information flow in a WSN with a) clustered architecture; b) non-clustered architecture.

resource-constrained nodes are used, introduce a second type of networks referred as *heterogeneous*. The motivation to use heterogeneous networks is that more complex hardware and extra battery energy can be embedded only in CH nodes, thereby reducing the hardware costs of the rest of the network [MR04]. However, fixing the cluster head nodes means that rotation of CH role is no longer possible. Consequently, there always exists a non-uniform energy drainage pattern in the network.

2.3 WSN Topologies

In WSNs, energy constraints dictate the need to reduce power of wireless transmitters resulting in small communication coverage of the sensor nodes. To deliver data packages to distances longer than the radio range of a single node, wireless sensor nodes cooperate employing multi-hop communication. Many nodes in the sensor network acts as repeaters, thereby reducing the link range coverage required and, in turn, the transmission power. Such repeating nodes are called routers or Full Function Devices (FFDs), while nodes without abilities to forward messages through the network are referred as measuring nodes or Reduced Function Devices (RFDs). There are two basic multi-hop network topologies, tree and mesh, as depicted in Figure 2.2.

2.3.1 Tree Topology

The tree network topology follows a hierarchical pattern, as illustrated in Figure 2.2a. Routers play the role of parents for other routers and measuring nodes in the next level of hierarchy. With tree topology, each node maintains a single route to the destination point. Hence, such a topology suffers from single point failure problem: If one router goes down, all routes relying on it become broken. Moreover, wireless networks with tree topology have other shortcomings. First of all, inefficiency of point to point communication is typical for tree-based networks: In cases where source and destination nodes are organized in two branches of the tree, data message is always forwarded via root of the tree. E.g., the route from the source node to the destination node in tree-based network illustrated in Figure 2.2a includes 4 hops, while mesh-based network with similar layout could provide 2 hop route, as can be seen in Figure 2.2b. Second, tree-based wireless networks do not scale well due to topology features imposing bottle-neck problem with increasing of network size. For instance, the root of the tree (Base Station in Figure 2.2a),

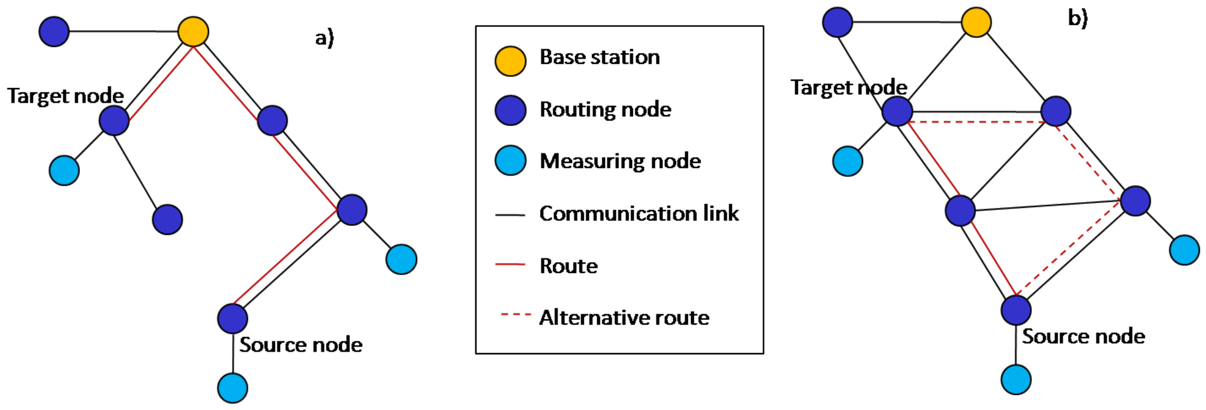


Figure 2.2: Multihop network topologies: a) Tree topology b) Mesh topology

will experience significant overload since it should forward all messages between nodes located in different tree branches. Third, addressing schemes applied in tree-based networks have a drawback of free network address space exhausting [CRM+07].

While featuring the number of disadvantages, tree-based networking does not require significant energy, memory and computational resources due to simplicity of routing protocols. Tree-based network topology can be effectively used for small or middle-size networks for applications which do not require high network reliability.

2.3.2 Mesh Topology

In mesh network, routers have communication link with all other routers in wireless range, as depicted in Figure 2.2b. Measuring nodes have single communication link with one of the selected routers in radio range, direct communication between two RFDs is not possible. The mesh network topology eliminates problems typical for tree topology. It provides redundant routes: Nodes maintain multiple routes to the destination point, so that if one router node goes down, the network automatically reroutes the data. For instance, a message from source node to destination node depicted in Figure 2.2b can be delivered either via shortest route consisting of two hopes, or via alternative route, if forwarding node in shortest route becomes unavailable. As result, mesh-based networks provide robust infrastructure for data flows and feature high system reliability.

Mesh-based networks also demonstrate better efficiency: Routing protocols search for the best available route to destination node. It is not always the shortest one: Depending on metrics used for route cost calculation, routing protocols might take into account quality of communication links and current workload of routers. Latter metric helps to avoid bottle-neck effect when network size is increasing. Hence, good efficiency and high scalability are features of networks with mesh topology. However, the price for improved abilities of mesh networking is complex routing protocols that require more energy and hardware resources available on the routers.

2.3.3 Mixed Topologies

Trade-off between functionality and costs can be achieved by mixing both tree and mesh network topologies. It can be done in networks with clustered architecture by using different topologies inside clusters and for communication among CHs. For instance, cluster heads could support mesh communication while inter-cluster communication may be tree-based.

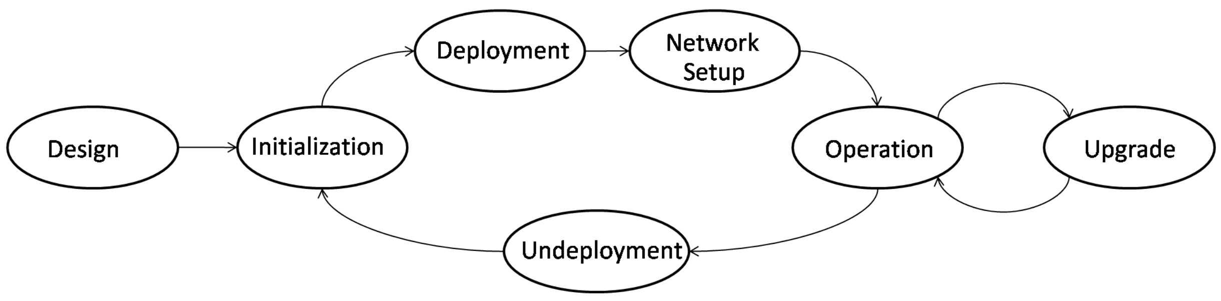


Figure 2.3: WSN Life Cycle

2.4 Life Cycle of Wireless Sensor Networks

WSNs with different topologies and architectures feature similar life cycle which can represented as shown in Figure 2.3.

Design In the design phase, the device nodes are produced and assembled.

Initialization In the initialization phase, device nodes are programmed according to the required WSN layout and purpose.

Deployment In the deployment phase, device nodes are deployed in the target environment.

Network Setup After deployment phase, on first activation, the WSN enters the network setup phase. This phase can be used by device nodes to discover communication peers, negotiate cryptographic keys and initialize any other services required for WSN operation.

Operation In this phase, the WSN is operational, i.e., the system works to fulfill its intended purpose within the target environment.

Upgrade Part or all of the WSN enter the upgrade phase if a modification of the deployed WSN interrupts normal operation. Examples for such modifications are adding hardware nodes or distributing software updates.

Undeployment In the redeployment phase, nodes are collected from the environment of the previous deployment.

Note that, since elimination of WSN components must be expected at any time during operational or upgrade phase (e.g., due to battery failure), there is no need to explicitly model a (partial) shutdown of the WSN. Similarly, the physical replacement of device nodes (due to malfunction or upgrade) can be modeled through the node elimination and subsequent upgrade of the WSN.

2.5 Security Objectives of Sensor Networks

In the following we provide a short description of the security objectives that are fundamental for every WSN. Security objectives address the protection of assets, i.e., security-critical objects of a WSN.

Integrity ensures that the asset is protected against unauthorized alternation (and refers to the ability to confirm that the asset has not been tampered with or modified).

Authenticity ensures that the asset is genuine, i.e., originates from legitimate entity.

Confidentiality ensures that the asset is protected against unauthorized access.

Availability ensures that the asset is always accessible for legitimate purposes.

Freshness ensures that no out-of-date version of the asset has been replayed.

Unclonability ensures that the asset cannot be copied.

Anonymity ensures unlinkability of the asset and its origin.

Security aspects of different applications differ, thus depending on particular application scenario security objectives may vary. Some application scenarios require to consider all security objectives listed here while security requirements of other applications may be fulfilled with the subset of these objectives.

3 Security Considerations (M1)

In this chapter, we consider the security aspects that are specific to the application scenarios targeted by the TeSOS project.

3.1 Problem Definition

This section describes the security-related aspects of the operational environment in which secure Wireless Sensor Networks (WSNs) are to be used. This includes the definition of the security-critical information to be processed in this context (Section 3.1.1), the assumptions made (Section 3.1.2), the adversary model (Section 3.1.3), and typical threats against WSNs (Section 3.1.4).

3.1.1 Assets

Assets in a WSN can be particular kinds of data in the network (data), general information about the WSN and its components and operation (meta-data), or certain devices and services of the WSN. The identification of assets is used to formulate typical threats against WSNs in Section 3.1.4 and to define the security objectives of a secure WSN in Section 3.2.

WSN.Application.Data Measurements of individual devices of the WSN or intermediate results stored locally for aggregation/processing within the WSN.

WSN.Location.Data Information of the physical location (coordinates) of a sensor or a subset of sensors of the WSN.

WSN.Control.Data Information that serves the general operation, synchronization, and signaling within the WSN. Examples of WSN control data are timing and interval data, power consumption management, routing information, and commands that induce specific actions of sensor nodes.

Sensor.Code All software and firmware that is stored on a sensor. Examples are the operating system of sensor nodes, custom applications, and software updates sent over the network.

Sensor.Metadata Metadata of employed hardware and software of a sensor node. Examples are node identifier, vendor, product and version numbers as well as information about software patch levels and employed algorithms.

Sensor.Configuration Software and firmware configuration of a sensor node. Examples are the configuration of operating system and provided services as well as parameters for key management and communication behavior.

We do not consider individual network nodes as primary assets since WSNs are typically designed to tolerate fault and even compromise of one or several network nodes. However, primary assets residing on a particular node can be affected when the node is attacked.

3.1.2 Assumptions

In the following, we document the assumptions that have been agreed upon for the TeSOS project. We describe the security aspects of the environment in which the secure WSN will be used or is intended to be used, including information about the intended usage, possible limitations of use, as well as physical, personnel, and connectivity aspects.

WSN.Devices The sensor network is heterogeneous, i.e., it consists of a single Base Station (BS) and two types of sensor nodes with different (hardware) capabilities, named Big Node (BN) and Small Node (SN).

WSN.Topology The sensor network features a hierarchical structure of clusters, where each cluster consists of one Big Node (BN) and several Small Node (SN). Basically, SNs communicate primarily with a specific BN, their Cluster Head (CH), and the BNs maintain communication with the Base Station (BS) and other clusters. Each cluster consists of a single BN and 5 to 50 SNs.

WSN.Routing The BNs communicate primarily with the BS using mesh-networking with other BN if required. Small Nodes (SNs) normally communicate directly with their designated Cluster Head (CH), i.e., in normal mode they employ one-hop communication, but mesh-networking is also supported as fall-back mechanism if the designated CH is not reachable.

WSN.Size The total number of devices in the WSN does not exceed $N = 1000$ nodes.

WSN.Deployment The location of each WSN node is known prior to deployment.

WSN.Location Each node is aware of its relative or absolute location in the WSN.

WSN.Dynamics TeSOS WSN is not dynamic, i.e., we assume that BNs and SNs have static positions after deployment unless manipulated by the adversary.

WSN.dataDelivery We assume periodic data delivery model with a single collection endpoint. Sensor nodes report measured data periodically to a single BS.

Nodes.Resources The BS has hardware resources compared to a Personal Digital Assistant (PDA)/Smart Phone. The BNs possess sufficient hardware resources to run multi-task operating system and regularly execute asymmetric cryptographic operations such as Elliptic Curve Cryptography (ECC), as well as state of the art symmetric cryptography. SNs have less computational resources and support only symmetric cryptography like hash functions, block ciphers, and stream ciphers.

Nodes.Hardware The hardware used in the WSN works according to its specification.

Nodes.Software The operating system and programs that run on the WSN nodes are implemented correctly. They are not malicious unless specified otherwise.

Nodes.Crypto The hardware used for the BNs optionally supports accelerated symmetric and asymmetric cryptography and a Pseudo-Random Number Generator (PRNG).

Nodes.PUF The hardware of BNs and SNs optionally also integrates a Physically Unclonable Function (PUF), a device-characteristic tamper-evident one-way function in hardware that cannot be simulated by the adversary.

3.1.3 Adversary Model

The adversary model defines the computational and storage capabilities of the adversary, i.e., the logical party that may want to violate security objectives of a WSN. These capabilities are used to derive the attack classes the adversary can launch. As proposed in [FB08], we represent adversary's capabilities as a set of values on the three dimensions *Intervention*, *Presence* and *Duration*:

Intervention defines attacks the adversary can apply.

1. *Passive outsider.* The adversary can eavesdrop on radio transmissions.
2. *Active outsider.* The adversary can eavesdrop on radio transmissions, but also inject bits in the channel and replay previously overheard packets.
3. *Resource constraint insider.* The adversary can eavesdrop on radio transmissions, inject bits in the channel and replay previously overheard packets. Moreover, an adversary is able to capture honest sensor nodes, tamper with their hardware and compromise cryptographic material.
4. *Resource powerful insider.* The adversary can eavesdrop on radio transmissions, inject bits in the channel and replay previously overheard packets. Also, an adversary is able to capture honest sensor nodes, tamper with their hardware and compromise cryptographic material. Moreover, an adversary can introduce malicious nodes which are more powerful in terms of energy and computational power than network nodes, e.g., lap-top class devices.

Presence defines the scale of attack and location where it is applied.

1. *Local.* The adversary affects a limited area of the network, e.g., radio transmissions within a limited radio range, or a small connected subset of sensors.
2. *Distributed.* The adversary affects several disjoint limited areas of the network, i.e., he can eavesdrop radio transmissions in different locations within a limited radio range, or he can affect small subsets of nodes distributed overall network. All adversarial nodes are able to communicate via out-of-band channels.
3. *Global.* The adversary affects all nodes in the network. All adversarial nodes are able to communicate via out-of-band channels.

Duration describes time available to the adversary for launching the attack.

1. *Short time.* The adversary has short time ad-hoc access to a WSN during it's operation excluding initial WSN deployment phase.
2. *Long time.* The adversary has long time access to a WSN during it's operation excluding initial WSN deployment phase.
3. *Life time.* The adversary has possibility to launch attacks as long as he needs and at any time, even during initial WSN deployment.

In context of TeSOS we consider a resource powerful insider with distributed presence and long time access to the network. Particularly, we consider an adversary with unlimited access to WSN during its operational phase, who can perform any attacks against sensor nodes and WSN communication links. The adversary can eavesdrop on radio transmissions, inject bits in the channel, and replay previously overheard packets. Also, an adversary is able to capture honest sensor nodes, tamper with their hardware and compromise cryptographic material. Moreover, the adversary can introduce malicious nodes which are more powerful in terms of energy and computational capabilities than legitimate sensor nodes. Multiple small connected subsets of adversarial nodes can be distributed overall network and are able to communicate via out-of-band channels and can use all compromised cryptographic secrets. However, we assume the following limitations on the adversary's capabilities:

ADV.OperationalPhase Physical security is provided during deployment, initialization and upgrade phase of the WSN such that an adversary can launch attacks only in the operational phase.

ADV.TrustedBS The BS cannot be compromised by the adversary.

3.1.4 Threats

In this section, we illustrate some typical WSN attacks or attack classes. We do not aim to present a complete list of attacks but only list the most typical.

For each described threat we (i) explain how threat can be mounted, (ii) provide examples and (iii) mention the security goals they violate. To fit to the general adversary model, the attacks described here are split into two groups: (i) Communication link intrusion and (ii) node intervention.

Attacks on Communication Link

Communication link attacks may affect all assets which are supposed to be transmitted over the network, i.e., WSN.Application.Data, WSN.Location.Data, WSN.Control.Data. Additionally, the asset Sensor.Code can be delivered to the nodes over-the-air during software update. Further, we will refer to the subset of these assets as "transmitting assets".

Link.Jamming An adversary may deliberately interfere with radio frequencies that WSN is using to interrupt a communication channel, e.g., in order to isolate parts of the network. This threat violates availability of transmitting assets.

Link.Eavesdropping An adversary may eavesdrop the transmitted data and analyze it, e.g., as preliminary step before routing attack is launched. Here the adversary needs to place wireless transmitter near to the targeted WSN node. Moreover, more sophisticated adversary can use high power transmitter and very sensitive receiver and monitor data flows in wide ranges. Eavesdropping violates confidentiality of transmitting assets.

Link.Data.Injection An adversary may inject forged data messages to the communication channel in order to affect data flow transmitted over the network. For example, the adversary can attack the routing protocol in order to disrupt routing, or he can flood the network with multiply forged data messages in order to exhaust energy resources of the network nodes taking part in forwarding and processing of these messages. This threat violates authenticity of the transmitting assets. Moreover, indirectly availability of these assets may be affected, e.g., in case if the routing is disrupted.

Link.Data.Replay An adversary may record eavesdropped messages and then replay them at later time. For example, some attacks on routing protocols can be launched by replying route messages, e.g., sinkhole attack [KW03] can be mounted by replaying "route advertise" messages of the BS. This threat violates freshness of the transmitting assets and may affect their availability (e.g., if the routing protocol is affected resulting in disrupted routes).

Link.Data.Modify An adversary may modify transmitting messages. Typical attack scenario requires the adversary to inject false node into a WSN. The false node may modify routed messages. This can result in false data reporting, in route disrupting or in replacing the legitimate node software with malicious code during software update. This threat violates integrity of the transmitting assets.

Link.Data.Relay An adversary may relay data messages in order to intercept communication between two legitimate nodes (so-called man-in-the-middle attack). One known example of man-in-the-middle attack on the routing protocol is the Neighbor attack [NN06], that makes two nodes that are in fact out of each other's communication range to believe that they can communicate directly. Neighbor attack can be applied in order to disrupt routing or to forward traffic selectively. This threat violates authenticity of the transmitting assets.

End-point Attacks

End-point attacks may affect all assets which are hosted by the nodes, i.e., Sensor.Metadata, Sensor.Code and Sensor.Configuration. Moreover, the asset WSN.Application.Data can be also affected due to it is typically produced or processed by the nodes.

Node.Destruction An adversary can physically destroy a node making a node unavailable, that will result in impossibility to perform a task the node is intended for, e.g., take and report measurements, aggregate and process data or route messages though the network. This threat violates availability of the WSN.Application.Data asset.

Node.Manipulation An adversary can access a network node and manipulate it. He can influence sensor readings (e.g., by heating a temperature sensor), screw out an antenna or simply remove batteries. Influenced sensor readings will result in reporting false data, while removing batteries or removing antenna will result in node unavailability. This threat violates integrity and availability of the WSN.Application.Data asset.

Node.Compromise An adversary may compromise one or more legitimate node(s) remotely, for example, by exploiting software vulnerability or by replacing the origin software with malicious code via over-the-air software update. When under control of the adversary, the node may access and modify data stored on the node, report false data, manipulate or misroute data being transmitted over WSN. This threat violates integrity of the assets Sensor.Code, Sensor.Metadata, Sensor.Configuration and WSN.Application.Data. In another attack scenario targeting similar goals, the adversary replaces the origin software with legitimate, but out-of-date software version which may have security critical vulnerabilities. Then the adversary may exploit such vulnerabilities and take control over the node. In the latter attack scenario freshness of the asset Sensor.Code is violated.

Node.Tampering A strong adversary may tamper with a node with the goal to extract sensitive information or manipulate the behavior of the node. For instance, he may observe power consumption and electromagnetic emissions of the analyzed device (side-channel attacks) or employ invasive tampering of internal node components (e.g., microprobing and UV lighting [Sko05]) to analyze or modify the internal node state. This threat violates confidentiality and integrity of the assets Sensor.Code, Sensor.Metadata, Sensor.Configuration and WSN.Application.Data.

3.2 Security Objectives

In the following, we describe the security objectives of the TeSOS WSN based on identified assets. The objectives are designed to cover all identified threats for the application scenarios considered in this study.

Confidentiality This property is required for the assets WSN.Application.Data as well as Sensor.Configuration. The assets WSN.Location.Data, WSN.Control.Data, Sensor.Code and Sensor.Metadata require confidentiality depending on the usage scenario.

Integrity This property is required for all defined assets. Assets WSN.Application.Data, WSN.Location.Data, WSN.Control.Data and Sensor.Code require integrity protection while in transit. Integrity of assets Sensor.Code, Sensor.Metadata and Sensor.Configuration should be ensured while they are hosted by the sensor node.

Authenticity This property is required for the assets WSN.Application.Data, WSN.Location.Data, WSN.Control.Data and Sensor.Code while they are in transit over the network.

Freshness This property is required for the assets WSN.Application.Data, WSN.Location.Data, WSN.Control.Data and Sensor.Code while they are in transit over the network and for the asset Sensor.Code while it is hosted by the node.

Availability This property is desirable for the assets WSN.Application.Data, WSN.Location.Data, WSN.Control.Data and Sensor.Code while they are in transit over the network. While full availability is hard to achieve, a certain resilience against Denial of Service (DoS)[1] attacks should be

[1]DoS attacks degrade or disrupt some capability or service in the WSN

provided, specifically with regards to (1) power consumption of individual nodes (secure wake-up and signaling) and robustness of the communication network against malicious interference (secure routing).

Table 3.1 illustrates the cross relation of the assets and considered security objectives. Security goals are represented by interception of the rows and columns in the table. We mark the security goals which should be met by the TeSOS system with '✓' and additional, use-case dependent security goals with '?'.

	Confidentiality	Integrity	Authenticity	Freshness	Availability
WSN.Application.Data	?	✓	✓	✓	✓
WSN.Location.Data	?	✓	✓	✓	✓
WSN.Control.Data	?	✓	✓	✓	✓
Sensor.Code	?	✓	✓	✓	✓
Sensor.Metadata	?	✓			
Sensor.Configuration	✓	✓			

Notation:
✓- required;
? - required depending on use-case;

Table 3.1: Security Goals: Intersections of Security Objectives and Assets

4 Related Work (M2)

In this chapter we provide an overview of the main related work on Wireless Sensor Networks (WSNs) security issues. We start with a detailed overview of known attacks and corresponding mitigation techniques or countermeasures in Section 4.1. Section 4.2 provides a comprehensive summary of literature on basic security mechanisms in sensor networks. We close the chapter with a summary of remaining problems with regards to the practical scenario focused in this study.

4.1 Attacks in WSNs and Countermeasures

Current WSN designs are vulnerable to attacks due to several reasons. First, wireless nature of node-to-node communication makes it easy to attack communication channels. An adversary may eavesdrop on data in transit or intercept communication unnoticed. Second, many application scenarios for WSNs assume that sensor nodes are deployed in hostile environments. When unattended, they are easy targets for physical attacks. Third, typical resource constrains with regard to power, memory and computational abilities make the task of securing WSNs very challenging. Fourth, many first generation WSN protocols were not designed with security in mind. Although they might be suitable for many applications, they cannot be used for security critical application scenarios. Finally, even in those WSN protocols which were designed as secure researchers discover vulnerabilities.

Possible attacks on WSN have been extensively studied during last years. Surveys [PS09], ontologies [ZMB08b] and taxonomies [HCGD06, RSS06, WS04] offer different approaches to classify these attacks. We distinguish endpoint and communication link attacks to comply with the adversary model introduced in Section 3.1.3.

4.1.1 Endpoint Attacks

Deployment of WSNs in hostile environments and typically, network nodes are not designed as tamper resistant to keep their costs low. Moreover, they can be deployed in hostile environments so that an adversary can easily access sensor nodes. These prerequisites make it easy for the adversary to launch physical attacks on sensor nodes. They can be destroyed, or their hardware can be attacked with the goal to compromise or replace software running on the nodes or to extract cryptographic material.

Node Destruction

Node destruction, a form of endpoint Denial of Service (DoS) attacks, poses a significant threat in WSNs that are deployed in hostile environment. The impact of this attack can be mitigated, i.e., by placing redundant nodes and camouflaging [RM08], but such a countermeasure can encounter only limited amount of destroyed nodes. Thus an alternative defense mechanism is desired such as node failure detection and reporting (e.g., by utilizing protocols proposed in [RCK$^+$05, TC05, RB06, SMLB07, MMSK08]. Typically, node failure detection is based on the idea of monitoring of sensor node liveness. Each node in the network sends heartbeat messages to their vicinity, thus node's disappearance can be detected. In general, a node can become unavailable if it runs out of energy, experiences program failure, or if an adversary performs endpoint attack and either destroys or captures the node[1]. To distinguish between energy exhaustion and possible attacker influence, the detection approach might be extended with reporting of battery voltage to predict failures induced by energy exhaustion [TC05]. To mitigate possible node capturing, key revocation mechanisms might be invoked to exclude suspicious nodes from the network, as proposed in [BBBD06].

Hardware Attacks on Sensor Nodes

Hardware attacks that involve structural modification of internal device components are typically referred as tampering [BBBD06], while attacks which consider circuit board level are referred as physical attacks [BBBD06], hardware attacks [Sko05] or node capturing [TP08].

Hardware attacks can be classified according to two criteria: behavior of the attacker and degree of invasiveness, as proposed in Ph.D. Thesis [Sko05]. The attacker can behave actively and passively. The active attacker influences general behavior of the device (e.g., tampers with internal device components), while the goal of passive attacker is to observe certain physical properties of the device in order to reveal security sensitive information (e.g., eavesdropping on the conductor wires connecting an external memory chip to a microcontroller). Degree of attack invasiveness can be categorized as following: Invasive, semi-invasive and non-invasive.

- **Invasive attacks.** Invasive attacks access the chip's internals. They require expert knowledge from the adversary and costly equipment. Examples of invasive attacks are reverse engineering and microprobing.
- **Non-invasive attacks.** Non-invasive attacks are relatively cheap and easy to conduct. Examples of non-invasive attacks are analysis of timing behavior or power consumption of a sensor node in order to reveal its cryptographic key (so-called side-channel attacks).
- **Semi-invasive attacks** are between noninvasive and invasive attacks. Examples of semi-invasive attacks are UV lightning and active photon probing (laser, X-rays). [Sko05] considers semi-invasive attacks as greater threat to hardware security, as they are almost as effective as invasive attacks but at lower cost.

The authors of [BBBD06] determine the actual cost and efforts needed to attack currently available sensor nodes. They show that hardware attacks are not so easy as it is usually assumed in the literature. Most of the attacks require significant amount of time to be launched and thus can be detected due to regular communication with neighboring nodes typical for WSNs. In contrast, results reported in [HBH05] show that very damaging attacks such as reading out content of Electrically Erasable Programmable Read-Only Memory (EEPROM), Flash and Static Random Access Memory (SRAM) memory can be performed fast enough to remain undetected ($<$ 1 minute). Note that SRAM is considered as safest place to store keys and other sensitive information due to its volatile nature. However, the ease with which the data can be extracted from SRAM (matter of seconds) motivates to look for alternative solutions.

Node capture attacks are formalized from the adversary perspective in [TP08]. Attacks are decomposed into sets of primitive events with a goal to understand their impact on the network protocols and security mechanisms. This paper do not propose a solution to defend against node capture attacks, but discuss the potential use of event-based decomposition and evaluation of metrics defined with respect to the decomposition for the developing suitable defense strategy.

Physical attacks described above differ from software-based endpoint attacks, which do not require physical access to sensor nodes to be launched.

[1]The work [BBBD06] shows that many hardware attacks result in temporary disappearance of the victim node from the network

Software Attacks on Sensor Nodes

Software attacks aim to exploit software vulnerabilities of the sensor nodes. In contrast to hardware attacks, software attacks may affect large number of nodes since many or all sensor nodes typically run the same software. As a result, the whole sensor network or a significant part of it can be compromised.

Limitations typical for WSN embedded software can simplify software attacks in some aspects, since operating systems for sensors (e.g., TinyOS [HSW+00b]) usually do not use memory protection and do not distinguish kernel mode or user mode execution. Though they are extremely challenging due to limited size of a single wireless message that imposes a hard limit into size of malicious code (e.g., 802.15.4 packet has size limit 128 bytes [IEE06]).

The first exploit for a wireless sensor node was presented in [Goo07] and extended in [Goo08, Goo09]. The author described stack overflow exploit mounted on sensor nodes based on MSP430 microcontroller. MSP430 has a Von Neumann architecture (VNA) that makes injected malicious code immediately executable. The authors of [GDKP10] introduced self-propagating worms in WSN nodes with VNA.

In contrast, infecting sensors with Harvard architecture (Harvard architecture) is considerably more complicated. Data containing in a malicious message can only be placed into data memory and hence cannot be executed. Work [GN08] introduces first attack on sensor nodes with Harvard architecture. The authors use a technique known as return-oriented programming (ROP) [One96] for execution of program code which is already presented in the sensor node. They exploit buffer-overflow vulnerability of the receiving function and invoke a subroutine which broadcasts content of the transceiver buffer while malicious packet resides in it. In this way, malicious packet propagates throw the network. This attack is transient and does not affect code memory of the infected node.

The first permanent code injection for sensor nodes with Harvard architecture was introduced in [FC08]. The attack targets Mica family sensor nodes based on ATmega128 microcontroller. It is shown that it is possible to inject code in the program memory due to the fact that in reality microcontrollers with Harvard architecture allow to write into the code memory under some circumstances, e.g., during software update procedure. If sensor node software contains code for firmware update, such instructions are presented in the memory and thus ROP technique can be applied to invoke them and to copy malicious code into code memory permanently.

Recommended preventive countermeasures against software attacks are selection of microcontrollers with Harvard architecture for sensor nodes [Goo07], heterogeneous network design and standard methods from software engineering to exclude code vulnerabilities [BBBD06]. Additionally, one may attempt to detect unauthorized software changes in the sensor. Known techniques to ensure software integrity are remote attestation and secure boot, we consider them in Section 4.2.4 and Section 4.2.2 respectively.

4.1.2 Communication Link Attacks

Communication link attacks are often classified according to network layers they are applied to (e.g., in [RSS06, ZMB08a]). We will follow a similar approach and split them into groups according to typical layers found in a sensor network stack: Physical, link, network and application.

Physical Layer Attacks

The physical layer defines the means of transmitting raw bits over a physical link connecting network nodes. The physical layer attacks target the transmission media of the communication link.

- **Jamming.** The objective of jamming attacks is to prevent sensor nodes from communicating by causing radio interference. In [XMTZ06] the authors survey different jamming attacks and distinguish four jamming strategies: *Constant jamming* emits continuous radio signal; *deceptive jamming* injects continuously regular packets to the channel; *random jamming* alternates between sleeping and jamming periods to save energy and *reactive jamming* transmits a radio signal only if activity on the channel is sensed. Constant, deceptive and reactive jamming strategies may cause the packet delivery ratio to fall to zero or almost zero, but are energy-inefficient. Random jamming is energy-efficient but less effective [XTZW05].

Countermeasures against jamming attacks include noise-resistant spread spectrum techniques and employment of error correcting codes to restore corrupted messages [NP03, Sta00]. Spread spectrum techniques, e.g., Frequency-Hopping Spread Spectrum (FHSS) and Direct Sequence Spread Spectrum (DSSS), transmit the signal over a wide bandwidth, much wider than the signal would require. The signal spread over the spectrum has high resilience to interference with narrow-band jamming. Unfortunately, transceivers employing sufficiently strong spreading techniques to resist jamming (such as FHSS) are too expensive to be used in most commodity WSNs. Error-correcting codes [Moo05] provide a mechanism to tolerate a certain level of corruption in messages, however, the error-correcting codes themselves also incur additional processing and communication overhead.

Reactive countermeasures against jamming include attack detection and defense actions. Jamming detection is typically based on statistics such as signal strength, carrier sensing time and packet delivery ration [XMTZ06]. When an attack is detected, the victim node may try to inform the rest of the network about the attack. It can either try to compete with the jamming signal by increasing transmission power or avoid it by changing frequency or physical location [XWTZ04]. Different from these strategies, [CCH07] proposes to tunnel messages from the jammed region to unaffected area via out-of-band communication channel (for instance, certain number of sensor nodes can be coupled with wired connection or be organized as frequency hopping pairs, e.g., using Bluetooth). When no affected by a jammer nodes are informed about presence of the attacker, the appropriate countermeasures can be undertaken. For instance, [WSS03] proposes to map out the jammed regions of a WSN, and to adjust routes in order to reduce the impact of the jamming on the whole network.

Link Layer Attacks

The link or Media Access Control (MAC) layer is responsible for node-to-node (hop-to-hop) frame delivery and provides channel arbitration for neighbors communication. Cooperative schemes that rely on carrier sense which let nodes detect if other nodes are transmitting face some security risks, particularly they are vulnerable to DoS attacks.

- **Link Layer Jamming.** Works [LHdHH05, LvHD^{+}05] introduce energy efficient link layer jamming attack. The core idea is to jam packets (similarly to reactive jamming), but to use sleep periods in between (as in random jamming). To make the attack more energy efficient than random jamming, knowledge of link layer protocols and probability distribution techniques are used to predict time of message appearance. The authors simulated the attack for link layer protocols Sensor MAC (S-MAC) [YHE02, YHE04], Lightweight MAC (LMAC) [vHH04] and Berkeley MAC (B-MAC) [PHC04]) and conclude that all of them are vulnerable, although LMAC seems to be more resistant than S-MAC and B-MAC. They also explore some potential countermeasures and conclude that all of them are not very effective.

- **Collisions.** In the collision attack, the adversary makes short time jamming of packets sent by legitimate nodes in order to corrupt the message [WS02]. It causes CRC error so that packet is retransmitted resulting in bandwidth loss and energy exhaustion of sensor nodes while the attacker is relatively energy-efficient. Error-correcting codes [Moo05] provide a flexible mechanism for tolerating variable levels of corruption in messages, however, malicious nodes can adopt to corrupt more data than the protocol can correct. Collision attacks can be detected by analyzing the packet delivery failure rate.

- **Denial of Sleep.** The denial-of-sleep attack targets energy resources of the WSN. Typically, WSN nodes have limited energy resources upon deployment. To conserve this critical resource, sensor nodes alternate periods of activity with low-power sleep mode in order to extend the network lifetime. Denial-of-sleep attacks force nodes to exhaust their energy and reduce the network life time significantly.

  Since the dominant source of power loss of sensor nodes is the radio transceiver, the attacker aims to keep the radio of the victim device awake, either in idle listening or in sending mode. Thus, defensive strategies typically aim to limit radio usage. As the data link layer is responsible for managing the radio, the protection mechanisms incorporated into link layer protocols are the most effective.

  The first study on denial-of-sleep attacks in WSN was introduced in [BGD05]. This work studies how the *broadcast denial-of-sleep attack* affects link layer protocols S-MAC [YHE02, YHE04], Timeout MAC (T-MAC) [vDL03] and B-MAC [PHC04]. In the broadcast attack, a single malicious node transits unauthenticated broadcast messages, forcing other nodes to receive them. Even if these broadcast messages are discarded upon reception due to authentication failure, the fact that all nodes stay awake to receive the messages has significant impact on network lifetime. The S-MAC protocol is shown to perform better under attack than T-MAC and B-MAC, but all of them are vulnerable to denial-of-sleep. Thus, a new MAC protocol is proposed for clustered WSNs, Gateway MAC (G-MAC), which performs better under the attack. In G-MAC only gateways (or cluster heads in our terminology) are vulnerable to broadcast attacks but nodes inside the cluster are not affected. To distribute the energy requirements among all sensors, G-MAC periodically elects new gateway nodes (and is thus limited to homogeneous WSNs).

  Studies [Ray08, RMBM06, RMBM09] discuss how jamming and the broadcast denial-of-sleep attacks influence power consumption. Additionally they introduce two new types of denial-of-sleep attacks: An *intelligent reply* attack and a *full domination* attack[2]. In the intelligent reply attack the attacker knows the MAC protocol and is able to distinguish control traffic from data traffic. Although details of the attack differ for different protocols, the main idea is similar: To delay the sleep period of the nodes by replying control messages. This keeps all the nodes awake until they run out of power. The full domination attack assumes presence of an internal attacker. Once the network is penetrated, all of the MAC protocols are susceptible to worst-case power consumption scenarios. In this case all nodes can be forced to remain continually in receive mode.

  The following defense strategies are proposed by the authors to counter these attacks: (i) link-layer authentication, (ii) anti-replay protection, (iii) jamming detection and mitigation, and (iv) broadcast attack defense. A novel approach to defend against broadcast denial-of-sleep attacks is proposed in [RM07]: A lightweight intrusion-detection mechanism employed at the MAC layer that classifies each incoming packet as either legitimate (meaning that it passes authentication and anti-replay checks) or malicious. Tracking the ratio of legitimate to malicious traffic, along with the percentage of time that the device is able to sleep, is used to identify a denial-of-sleep broadcast attack.

- **Acknowledgment Spoofing.** Several sensor network routing algorithms rely on implicit or explicit link layer acknowledgments [KW03]. Due to the use of broadcast media, an

[2]particularly protocols S-MAC, T-MAC, B-MAC and G-MAC are considered

adversary can spoof link layer acknowledgments for overheard packets addressed to neighboring nodes. Attack goals include convincing the sender that a bad quality link is good or that a dead or disabled node is alive. Such manipulations can be used to affect network routes which may be selected in accordance with link reliability reported by the link layer. This vulnerability can be addressed by ensuring that link layer data is encrypted and authenticated.

Routing Layer Attacks

The routing layer in WSNs is responsible for end-to-end packet delivery through intermediate sensor nodes. Most attacks on the routing layer require an adversary who penetrated the network (i.e., a malicious node presented in the network). However, attacks can be launched by an external adversary even if all traffic in the network authenticated, encrypted and replay protected.

Generally, the attacker can have the following goals: (i) to insert a malicious node into the route with a goal to launch other attacks, (ii) to eavesdrop or manipulate forwarded messages, (iii) to attract data generated by legitimate nodes, e.g., for analyzing them and (iii) to disrupt routing with the goal of DoS.

- **Routing State Corruption.** Attackers are able to manipulate routing tables stored on sensor nodes by spoofing, altering, or replaying routing information in transit by creating routing loops, attracting or redirecting network traffic, extending source routes, increasing end-to-end delay, etc.

  Message authentication and replay protection can defeat these attacks in case of outside attackers. Preventive defense may include usage of stateless protocols like IGF [BHSS03] which make forwarding decisions "on-the-fly" and do not require routing tables to be stored on the node.

- **Bad-Mouthing Attack.** Some routing protocols may use reputation-based systems to find a most trustworthy candidate to forward message. Second hand evidence can be used to judge the trustworthiness of a node [LRAFG10]. In a bad-mouthing attack, a malicious node votes against honest member reducing its chances to be selected.

  To mitigate bad-mouthing attacks, second hand reports can be ignored or handled with less priority. Depending on the reputation system, it is possible to reliably detect the adversary as its reports are outliers within a larger set of reports about a specific node. In this case, the malicious node can be excluded from the system or assigned a lower trust value to mitigate the impact of its attack [HZR09].

- **Rushing Attack.** The rushing attack was introduced in [HPJ03b]. The goal of the attacker is to increase the probability for the malicious node to invade the route. Rushing can take place in some demand-driven routing protocols such as tinyAODV [Tin07] and TinyHop [SCHM08] which use the duplicate suppression mechanism for their operation. Duplicate suppression mechanism consist in forwarding the first received Route Discovery packet and ignoring any duplicates arriving at later time. To launch the attack, an attacker quickly propagates further received Route Discovery packets, ignoring specified protocol delays. Alternatively, the attacker can use a powerful transmitter to deliver the packet to destination within more than one hop at once. In both cases, such Route Discovery packet has higher probability to reach its destination faster.

  [HPJ03b] proposes three components for defense against the rushing attack: (i) secure neighbor detection (employs a reply protected mutual-authentication protocol which also takes into account response delays, (ii) a secure route delegation where each node should decide, which neighbors will forward its route-request, (iii) a randomized route-request forwarding where the selection of the Route Request message to forward is randomized.

- **Wormhole Attack.** Another effective way to insert a malicious node into a route is creating a *wormhole* [HPJ03a].

  To mount a wormhole attack, the adversary involves two distant malicious nodes connected with a high-speed channel, e.g., laptop-based nodes equipped with WiFi. Messages received in one part of the network are tunneled via the low-delay channel and replayed at the other end. If one malicious node is located near to the base station and forwards routing updates to the second malicious node through the fast channel, i.e., wormholes them, such packets will reach the targeted area considerably faster and provide a high-quality route to the base station. As result, all traffic in the surrounding area will be routed via malicious node since alternate routes are less attractive. Note that the attack will succeed even if route update message is encrypted, authenticated and replay protected.

  Preventive countermeasures against wormholes include the selection of geographic routing protocols [KK00, Kar05], in which the next forwarding node is selected according to its geographical location. Geographic routing protocols require deployment knowledge which can be provided either manually during network setup, or can be obtained using localization algorithms [LP04, XOL+07]. Alternative approaches follow the detection strategy. They rely on observation of the symptoms induced by wormholes, e.g., localization anomalies [DFN06], distortions of substructures on network connectivity graph [MGD07, ZMB08a], inconsistencies of network layout (i.e., network connectivity graph and distance between neighbor nodes in a graph) [WB04, DLL+09], abnormal variations of length of the routes [LB05, RGCL09], mismatches of neighborhood observations [LB05] or mismatches between the key used to sign a message and the message radio fingerprint [RC07].

  Many existing solutions make specific assumptions on the network such as known node localization [DFN06, RGCL09], presence of special anchor nodes [RGCL09] or specific communication models [MGD07]. Detection algorithms proposed in [WB04, BDV05, RC07] are centralized and thus do not scale well. The most recent distributed detection approach is introduced in [DLL+09]. It claims not to impose any rigorous requirements and assumptions on the network, has nearly 100% detection rate and does not cause false positives. Although promising, the computational and communication overhead imposed by this algorithm is not yet analyzed.

Once the malicious node invades the route, the adversary can analyze, filter, drop, alter, reply or delay forwarding messages.

- **Jelly-Fish Attack.** If the attacker delays forwarded messages in order to decrease the performance of real-time applications, such an attack is referred to as *jelly-fish* attack [AHK04]. As the jelly-fish attack does not lead to unavailability of services provided by the network but only decreases performance, this attack got a little attention of researchers: No mechanisms are designed to defend against jelly-fish attack. It appears that detection and diagnosis of this attack is too costly for resource constraint WSNs since it maintains compliance with protocols.

- **Black-hole and gray-hole attacks.** The attacker may be interested in dropping forwarded messages rather than delay them. If the attacker silently drops all received packets, such an attack is known as a *black-hole attack*. Very sophisticated adversary can drop packets selectively and forward correctly the remaining traffic in order to avoid raising suspicions. Such variation of attack is named *selective forwarding* or *gray-hole attack*. More powerful variations of black hole and gray-hole attacks occur when several compromised nodes cooperate in order to launch the so-called *cooperative black or gray-hole attack*. These attacks were first discovered and studied in the context of ad-hoc wireless networks [RFS+03, AGD08, Ban08, SA08a, SA08b].

In general, defense strategies against the selective forwarding attack in WSNs include multi-path routing, attack detection or a mixture of both approaches. Multi-path routing protocols like [GGSE01, DQW03, MLWSW07] enhance the reliability of the data transmission in presence of the attacker, especially if disjoint paths are discovered or multiple base stations are available. A variety of mechanisms are developed to detect and localize the attacker in the network. An Intrusion Detection System (IDS) proposed in [YX06] and improved in [XYG07] uses multi-hop acknowledgments to detect routing misbehavior. REWARD scheme [Kar05] employs neighbor monitoring to detect nodes which fail to forward received messages. To detect cooperative malicious nodes, REWARD proposes to forward messages with increased power such that two-hop neighbor monitoring is possible. The solution offered in [LC06] utilizes secure two-hop neighbor discovery and one-hop neighbor monitoring. The mechanism proposed in [KSMS07] is carried out entirely by the base station and utilizes a classification method based on support vector machines (SVMs)[3] to analyze routing information local to the base station: The hop count and bandwidth. Scheme [DSWC09] applies watermarking to detect dropped messages and to localize suspicious nodes, and manages trust values which reflect the trust credit of each sensor node in the network. The Dynamic Trust Management System (DTMS) [RSCD08] also associates trust with values: It assigns each node in the network with a trust vector consisting of three parameters. Apart from selective forwarding attacks, DTMS is able to detect sinkholes (described below).

In recent research, there is a trend to design IDS which are able to detect two and more attacks in WSN rather than to defend against a specific attack. We will discuss such systems in Section 4.2.7.

- **Sinkhole Attack.** In a sinkhole attack [KW03, KGD08a], a malicious node tries to attract all traffic from the surrounding network by making the node look very attractive to the nearby nodes with respect to the routing metric. The sinkhole attack can be launched quite effectively in networks with aggregation routes, where measured data is aggregated from multiple sources to a single Base Station (BS). In this case, a malicious node tries to convince the neighboring nodes it has the best route to the BS. For instance, it can spoof or replay an advertisement from the BS with an extremely high quality route metric. To control all traffic in the network, the adversary needs only to launch the sinkhole attack as close as possible to the BS. In this case neighboring nodes will choose a malicious node as the best candidate to forward messages, and all the traffic coming from them will end up in the sinkhole. So the attack can be very effective even when launched by a single malicious node.

  Authenticated and replay protected communication helps to prevent an external attacker from launching the sinkhole attack. But these countermeasures cannot defeat against internal adversary (i.e., the adversary who compromised a number of nodes and retrieved their cryptographic material). The approach presented in [PFVS08, PFVS09] proposes two topology-based reconfiguration protocols which are resilient to sinkhole attacks. Unfortunately, they are limited to only networks with tree-based routing topologies. All other solutions [NLL06, KDGM07, NLL07, MC09][4] rely on an IDS for sinkhole detection.

- **Sybil Attack.** The Sybil attack was first described in [Dou02] in the context of peer-to-peer networks. In a Sybil attack, a malicious node pretends to be multiple nodes by claiming multiple node identities. This attack can affect routing mechanisms in sensor networks , fair resource allocation , voting algorithms and data aggregation.

  Following major approaches exist to defeat against the Sybil attack: (i) cryptographic

[3]SVMs are a class of machine learning algorithms [CV95]
[4]IDS presented in [KDGM07] also detects black-hole and selective forwarding attacks

approach, (ii) identity registration, (iii) remote software attestation (see Section 4.2.4) and (iv) attack detection.

The cryptographic approach relies on associating the node identity with the keys assigned to the node. For instance, the node identity can be associated with the verifiable public key certificates, or with key-related information preloaded into each sensor node in key predistribution schemes (see Section 4.2.1 and Section 4.2.1). These keys or certificates can be validated by other nodes in the network. Validation can be performed either locally, by single nodes independent of other nodes, or globally, via collaboration of many nodes.

Identity registration can prevent the Sybil attack, since any node can check by asking a central authority for a list of legal identities to validate another node as legitimate. The disadvantage of this approach is its centralized nature.

Remote software attestation (see Section 4.2.4) can be employed to prevent Sybil attack, since the code running on a malicious node must be different from that on a legitimate node.

Many approaches have been proposed by researchers to detect the Sybil attack. The radio resource testing as proposed in [NSSP04] is based on the assumption that an attacker cannot use one device to send on multiple channels simultaneously. Location based solutions [SM06, SM09] check that no identities are at the same position. They require deployment knowledge and suitable only for static networks. Methods proposed in [DS06, WYSC07, AMP08, LWZZ08] make use of the Received Signal Strength Indicator (RSSI) to infer the distance between two identities and further determines the positions by use of the RSSI information from multiple neighbor nodes. They are also suitable in static networks only. The solution proposed in [WLZC08] utilizes Time Difference of Arrival (TDoA) method to detect Sybil nodes. It requires time synchronization in the network and uses the so-called beacon nodes with known locations which overhear the network traffic looking for messages with the same TDoA ratio but different identities. The algorithms proposed in [SWC09, WSC10] are based on neighbor knowledge of each node and do not rely on knowledge of deployment.

- **Neighbor Attack.** A neighbor attack can be launched by an internal attacker: Malicious node can replay control packets without updating them as required by the routing protocol. For instance, in many-to-one/source routing protocol included into ZigBee (ZigBee) networking stack the forwarding node must record its Id in the packet before forwarding the packet to the next node. Omitting this operation makes upstream and downstream neighbors to wrongly believe that they are within communication range of each other, resulting in a disrupted route.

  Countermeasures against neighbor attack include applying link layer security framework to defend against an external attacker. To resist an internal adversary, mechanisms to detect black and gray hole attacks can be applied, since breaking a route at a forwarding node can be considered as dropping data packets at that node in a black-hole attack [NN06].

- **Hello Flood Attack.** Many routing algorithms require nodes to broadcast special packets, so-called hello messages, to announce themselves to their neighbors. Nodes receiving such a packet may assume that they are within regular radio range of the sender. In the hello flood attack [KW03], the attacker transmits spoofed or overheard hello packets using a powerful radio transmitter, announcing false neighbor status to the network. This way, legitimate nodes will attempt transmission to the malicious node, although many of them are out of its radio range.

  The attack is possible even if the attacker has no sensitive receiver, since broadcast packets do not typically require acknowledges. Thus, authors of [HMORH06b, HMORH06a] offer to verify link bi-directionality between neighbor nodes to mitigate the attack. To resist an attacker possessing sensitive receiver, they propose multi-path multi-base station routing protocol which increases probability for data to be delivered in presence of the attacker.

The author of [Kho08] proposes detection mechanism as countermeasure. In the proposed scheme a few randomly selected nodes cast a vote to the base station for each hello packet received and the base station validates the legitimacy of the hello request. Voting nodes are periodically reselected by turn rolling algorithm. This solution is limited to WSNs where all nodes are aware of their geographical region.

A preventive countermeasure is to utilize network layer protocols which do not rely on HELLO packets. For example, stateless routing protocols like IGF [BHSS03] make forwarding decisions independently hop by hop.

Application Layer Attacks

The application layer deals with data produced and processed by the network. In application layer attacks the adversary typically injects forged data packets in order to influence (aggregated) sensor readings or with a goal of DoS. Combining packet authentication and anti replay protection mitigate these attacks, but these countermeasures are less effective if the network has compromised nodes.

- **Data Spoofing.** An adversary can inject crafted data packets with a goal, e.g., to arise false alarms or to influence results of aggregated and processed sensor readings. Secure aggregation techniques (e.g., [Wag04]) help to tolerate a small number of false sensor observations. A second direction for resilience to false sensor observations is to gather multiple, redundant views of the environment and crosscheck them for consistency [PSW04]. Meanwhile, when many data values are collected, a histogram may be constructed; extreme outliers may indicate malicious spoofed data.

- **Data Flooding.** In a data flooding attack, an attacker injects multiply forged or replayed data packets into the network. The goal of the attacker is to consume network resources such as bandwidth and energy by forcing network nodes to route messages to a base station. This attack is especially effective in a context of WSNs with a tree topology (see Section 2.3.1. This particular variation of the attack got a name *Path-based DoS (PDoS)* attack and was studied in works [DHM05, DHM06a]. A PDoS attack not only exhausts nodes along the path, but also exploits the tree-structured routing resulting in inability of leaf nodes to communicate with the base station. Rate-limiting[5] can mitigate the impact of the attack. Work [DHM05] propose a high resilient to node compromise solution based one-way hash chains to enable PDoS attack detection by each intermediate node.

- **Overwhelm Attack.** In an overwhelm attack [RM08], an attacker injects forged or replayed query messages, as opposite to data flooding attack, where data messages are injected. The overwhelm attack is more powerful since a single query message sent broadcast over the network force multiply sensors to report sensor measurements. This cause the network to forward a large volume of traffic to a base station consuming network bandwidth and draining nodes energy. This attack cannot take place if sensor measurements are reported at fixed intervals and are not driven by data query requests. Rate-limiting and data-aggregation algorithms can reduce attacks' effects by reducing a volume of network traffic.

[5]Limiting the number of packets an intermediate node can forward per second

4.2 Security Mechanisms in WSNs

Current research on security mechanisms in WSNs can be divided into four categories [RZL05]: (i) Key management; (ii) Secure routing; (iii) Specialized security services such as attestation, secure wake up, secure time synchronization and trust management; (iv) Intrusion Detection Systems (IDSs). In the following we provide an overview of related work on these topics.

4.2.1 Secure Key Management

Key management is the provisioning of cryptographic keys to low-level cryptographic algorithms for, e.g., authentication or encryption of data. Two basic cryptographic schemes must be distinguished, symmetric and asymmetric schemes.

To reduce the impact of key compromise, symmetric key schemes require a separate key for each communication channel between a set of parties. The number of keys to manage thus increases quickly with the number of involved parties and channels. Further, the fact that a key is shared between multiple parties reduces the robustness of the overall system towards compromise of individual parties. On the other hand, cryptographic operations on asymmetric keys typically involve much higher computational costs than operations using symmetric keys. Key management systems thus typically combine different techniques to meet security as well as performance requirements of practical systems. Forward security, i.e., the protection of past communication against key compromise in the future, is another important security objective of such systems.

In case of WSNs, the performance impact of different key management schemes is of particular importance. As such, these systems focus less on the management and negotiation of keys and more on a clever pre-distribution prior to deployment that allows efficient yet secure communication. Further, due to the inherent and much higher computational costs, WSN literature mainly explored symmetric key management and only recently considered the use of asymmetric schemes. Regarding performance considerations, we lend the following requirements from the comprehensive survey in [cY05]:

- **Storage.** The amount of data storage required on the individual node to save different types of keys and parameters.
- **Computation.** The amount of computation required for setting up individual communication channels.
- **Communication.** The amount of communication required to to set up an individual communication channel.
- **Scalability.** The ability of the scheme to scale to any desired network size, particularly large sizes. Also, the support for upgrade (i.e., enlargement) of the network after initial deployment.
- **Connectivity.** Probability that two parties can generate or find a shared key.
- **Resilience.** The amount of nodes in the system that can be compromised without affecting the security of communication channels of other parties.

While several key distribution systems have been proposed, most of them are designed for specific WSN architectures and deployment scenarios. None of them fit the possibilities and requirements of the TeSOS scenario, i.e. the use of Physically Unclonable Functions (PUFs) in a hierarchical, rather small scale network with possible hardware security mechanisms and hardware acceleration on the Cluster Heads (CHs). We thus provide a general overview of available ideas and architectures that might be used for the key management in TeSOS. For a more detailed description and comprehensive performance comparison of key distribution schemes published until 2005, see [cY05].

The two trivial extremes for key distribution in WSNs are to either provide a dedicated key for each possible communication link or to use a single master key for all communication links between all nodes. The first solution provides maximum resilience but requires each node to store $N-1$ keys and corresponding node IDs, where N is the WSN size, i.e., the number of devices in the WSN. The second solution requires minimal storage but provides no resilience against compromise. In the following three sections we present the several trade-offs between these two extremes that have been proposed in literature. We distinguish (i) pairwise key management schemes, (ii) hierarchical and group-wise schemes and (iii) general enhancements to symmetric key management schemes. We close with a review of current work on asymmetric key management.

Deterministic Pair-wise Schemes

The work in [LN03c] proposes to use knowledge about the physical topology to provide only the fraction p of keys that are actually reachable and that the individual node is desired to communicate with. The storage complexity is reduced to $2pN+1$ to save the peer keys together with the corresponding node ID plus the node's own secret key. Another approach is to interpret the network as a graph that is composed of star-like sub-graphs, as proposed in [LS05b]. Nodes are then provided with an individual master key and a link key derived from the master key of the respective root node. As a result, any node is either root or leaf node towards another node and can thus either generate a link key or use a stored derived key. The storage complexity for each node is reduced to $2r+1$ for keys and their corresponding key IDs as well as the node's own key, where r is the number of children of the respective node. Work [LS05b] also offers to extend the scalability of the scheme by grouping l sensor nodes into a single node, i.e., using each master key for l nodes simultaneously and thus sacrificing resilience.

Using *Combinatorial Design Theory* as proposed in [cY04, LS05a], key distribution can be mathematically designed such that in a network of size n^2+n+1, n^2+n+1 keys are distributed to a set of $n+1$ nodes such that after deployment, each pair of nodes has exactly one common key in its key chain. As an example, for $n=2$ and $N=7$, the keys $K_1, K_2, K_3, K_4, K_5, K_6, K_7$ can be arranged into 7 key chains as follows: (K_1, K_2, K_3), (K_1, K_4, K_5), (K_1, K_6, K_7), (K_2, K_4, K_6), (K_2, K_5, K_6), (K_3, K_4, K_7), (K_3, K_5, K_6).

In this scheme, n must be a prime power and the probability that a link is compromised when a node is captured is $\frac{1}{n+1}$. The authors of [cY04] also propose an extension such that, while not every pair has a common key, there are is always a third node that can be used to establish a path between them. Naturally, this requires that such a third party node is within the reachable neighborhood and thus results only in a probabilistic key connectivity. To address the restriction of n being a prime power, [cY04] also describes a combination of this approach with a pair-wise probabilistic key distribution scheme.

The key matrix-based scheme first proposed in [Blo85] was adapted for WSNs in [DDHV03, LS05b]. A matrix-based key distribution scheme interprets the set of all possible pair-wise link keys of a WSN as elements of a symmetric $N \times N$ matrix K. The matrix is then divided into a public key matrix G and private key matrix D such that $K = (D*G)^T * G$. The individual node n_i of the WSN stores row i of the private and column i of the public key matrix. To compute a link key, two nodes n_i and n_j exchange their public key data and each derive the corresponding element K_{ij} or K_{ji} of the key matrix K. Since K is symmetric, $K_{ij} = K_{ji}$ is the shared link key. Link keys of uncompromised nodes are secure while less than λ nodes are compromised, where λ is a linear factor in the size of the generated public and private key matrices. The scheme is further generalized in [DDHV03], where the authors create multiple private matrices D_r and give a random selection of p such matrices to each node, trading key connectivity for resilience.

In a similar approach, [LS05b] deterministically divides the set of nodes into partitions such that nodes of different partitions can not establish a direct link key. Resilience is increased since now a coalition of λ nodes from the same partition is required to compromise all other link keys.

An approach based on a polynomials was proposed in [BDSV+98]. In this scheme, a symmetrical polynomial $P(x, y) = P(y, x)$ with coefficients from $GF(q)$ for sufficiently large prime q is used to derive pair-wise link keys. Each node n_i stores a localized version $f_i(x) = P(i, x)$ of the polynomial. For node n_i to derive the common key K_{ij} with node n_j, it simply evaluates $f_i(j)$. The number λ of coefficients of P determines the resilience of the scheme in the sense that a coalition of $\lambda+1$ nodes is needed to derive all link keys. Similar to previous trade-off approaches, [LN03b] proposes to partition the WSN, distributing multiple polynomials and thus sacrificing key connectivity for scalability and/or resilience, since polynomials can be smaller but some node pairs may not have a shared polynomial to derive a common key from. The polynomials to store on each node can be selected randomly, in a grouped fashion to reduce communication overhead or based on available deployment topology information as proposed in [LN03c].

Probabilistic Pair-wise Schemes

The amount of required key storage for pre-distributed pair-wise keys is reduced in [EG02, CPS03] by storing only $N * p$ keys, where $0 \leq p \leq 1$ is the probability that any two nodes will be able to communicate in the resulting system. Another approach to modify the fraction of keys known to each node is presented in [DDH+04], where an approximate knowledge of the later physical topology is assumed and keys are distributed where the likelihood of reachability is large.

The authors of [ZSJ03] assume that the adversary needs a minimum time t to compromise a node. In the deployment or upgrade phase, they deploy a node A with a master key K_m, two keyed hash functions $f_x(\cdot)$ and $h_x(\cdot)$ where x is a node identity string, and a derived device key $K_A = h_A(K_m)$.

Within the time frame t, A can establish a pair-wise key K_{AB} with any discovered node B using $K_{AB} = f_{K_B}(A)$, since B still knows its derived key K_B and A can derive it as $K_B = h_B(K_m)$. After time t, key K_m is deleted but K_A is kept. As a result, only new nodes with knowledge of K_m can establish new links with node A. To establish keys with sleeping nodes that do not wake up until t, [ZSJ03] suggests that A asks available nodes for a list of node IDs within the area and derives the corresponding link keys in advance. Key connectivity and resilience of this scheme are probabilistic, but can be very good if t is chosen correctly. However, results on residual memory traces [HSH+08] and the sensor node's very limited resistance to physical attacks [HBH05] limit the practicality of this solution.

Enhancements to Pair-wise Schemes

Several protocols require an additional key discovery phase after deployment to determine what peers are reachable, whether link keys can be established with them and what other nodes can be reached via these links. In this phase, typically each node broadcasts the list of key IDs from its key chain and observes the corresponding announcements of any neighbors. The communication load in this phase can be reduced by grouping key IDs such that all IDs in a group are known from the corresponding group ID [HLV04] or, in probabilistic key distribution, by using a Pseudo-Random Function (PRF) to generate the list of IDs known to each node [ZSJ03] and only transmitting the respective PRF seed.

The storage complexity can be further reduced by grouping l link keys into a single localized master key. This reduces resilience as any compromised node will compromise the communication link security of $l - 1$ other nodes, however, the required storage is also reduced by a factor of l [LS05b]. A similar idea is described in [DCL04], where multiple master keys are used in combination with random nonces and a PRF to reduce storage. Each generation of nodes uses its own master key as well as derived keys of the master keys of future (anticipated) generations.

However, while such a scheme is useful in WSNs that are upgraded frequently, the resilience against compromise of a single generation of nodes is very low.

To improve the resilience of link keys, multiple *key enforcement* mechanisms have been proposed. The work in [CPS03] proposes to demand q common keys between a pair of nodes in the key discovery phase to derive a composite link key using a one-way function with the q keys as input. Unfortunately, this approach also has a direct impact on key connectivity. This is mitigated in [CPS03, ZSJ03] by sending additional key shares through alternative already established communication paths towards the target node or by letting cooperative nodes provide this information based on already established link keys [DPMM03]. While key reinforcement techniques can achieve higher key resilience, an adversary who recovers the initial link keys of a node might be able to deduce a reinforced key based on recorded communication. In general, the lack of forward security is a serious problem for any of the symmetric key distribution schemes.

Hierarchical and Group Key Schemes

Communication in hierarchical WSNs is typically structured in a tree-like graph with more robust or powerful sensor nodes close to the root of the tree. As a result it may be affordable to deploy dedicated pair-wise symmetric keys for each leaf to communicate with its respective adjacent root node. Root nodes can then either be trusted to forward messages directly, transport random end-to-end session keys or assist in establishing a shared secret [AUJP04, CCWW00, ZSJ03, LCEH03]. The authors of [ZSJ03] emphasize how battery use can be reduced if sensor nodes can listen to communication of their peers, e.g., to prevent transmission of redundant sensor readings. They suggest the use of group or even network-wide keys so that neighbors can decrypt such messages as well.

A simple approach to establish group keys is to leverage a pair-wise key distribution scheme. A group key can then be chosen by one of the group members and transported to all others [DCL04, ZSJ03]. In [BDSV$^+$98], each group member chooses a random key share and sends it to all other members. Each member then computes the group key as a function of the received key shares. The latter approach is slightly more robust as there is no single party that choses the group key but requires a significant communication overhead. Another scheme proposed in [BDSV$^+$98] is an extension of the polynomial key distribution scheme from Section 4.2.1. It uses a polynomial with l arguments instead of two, thus allowing up to l nodes to establish a shared link key.

The μ-TESLA authenticated broadcast scheme presented in [PSW$^+$01] uses a delayed key disclosure and a hash-key chain instead of asymmetric mechanisms. Messages are send in fixed time intervals known to sender and receiver. A broadcast message of interval i, where $0 < i < n$ is authenticated with a symmetric key K_i disclosed later in interval $i+1$. The keys K_i are taken from from a hash chain such that $K_i = f(K_{i+1})$. Nodes are pre-supplied with an authenticated key K_0 and thus able to authenticate successive disclosures of K_i since $K_0 = f^i(K_i)$. The scheme is robust to transmit errors as lost keys can be recovered once later keys are disclosed, however, it requires that sender and receiver are loosely time-coupled or a receiver might accept a message whose authentication key was already disclosed to the network. As receivers must cache broadcast data until it can be verified in the next time interval, μ-TESLA introduces a potential DoS vulnerability where an adversary may flood nodes with spoofed messages. Intervals can be shortened to reduce the impact of this attack but this will result in increased overhead for publishing and computing authentication keys. Work [SNW06] thus proposes to use a short interval for sending data followed by a long interval where the corresponding authentication key is sent.

μ-TESLA is used in [ZSJ03] to authenticate a group encryption key in advance before relying on the individual nodes to distribute it.

The authors of [BT03] modify μ-TESLA into a certificate revocation scheme where the base station resembles a Certificate Authority (CA) that discloses expired certificate keys. Other modifications are presented in [LN03a, LN04] to reduce the communication load for initial μ-

TESLA bootstrapping.

Asymmetric Schemes

Due to the disadvantages of symmetric key cryptography, the suitability of asymmetric key cryptography for sensor networks received increasing consideration in recent years.

The first general evaluation of RSA and Elliptic Curve Cryptography (ECC) for sensors with 8bit Atmel ATmega128 and ChipCon CC1010 processors concluded that asymmetric cryptography is expensive but generally possible [GPW$^+$04]. They review and implement common optimization techniques like the use of Mersenne Primes and projective coordinate systems and conclude that ECC is more suitable for WSN because of the potentially lower key size and better scalability for processors with small word size [GPW$^+$04].

At the same time, an RSA-based protocol suite was proposed and implemented in [WKC$^+$04] with the general result that even RSA is possible on the Crossbow MICA2, although each use considerably drains the battery. Another proposal in [HCK$^+$03] is to use a hybrid scheme where one of the peers, e.g., a Cluster Head (CH), is sufficiently powerful to carry out asymmetric computations.

ECC was later implemented in [MWS08, LN08] for TinyOS (see Section 7.4). Both works evaluate computation time, memory usage and energy consumption for several sensor platforms and conclude that ECC-based key management is a viable solution, e.g., for initial key establishment after deployment. With the evaluated hardware however, energy consumption and computation time are too high for regular use of ECC. According to the authors, depending on implementation and platform, a node's battery can be consumed within 50 ECC operations.

Works [EAA$^+$06, AQ05, AQR07] leverage a highly efficient authenticated DH protocol presented informally and without security proof in [Ara99]. Work [EAA$^+$06] presents a pairwise key agreement with the option to offload one of the exponentiations to another party while [AQ05] simply uses the pairwise key agreement to circulate a common symmetric key in a group. Work [AQR07] uses short RSA public key exponents and RSA key transport to impose a significantly higher computational burden on the requester of a connection, thus mitigating DoS attacks by compromised WSN nodes. However, the use of RSA generally drains the battery about twice as fast as ECC and the work does not consider stronger adversaries with special purpose hardware or laptops.

The authors of [ODL$^+$07] point out that Identity-based Encryption (IBE) is well suited to WSNs. Using IBE, nodes can identify themselves using their serial number or other strings like in symmetric key management, yet only a few critical global key parameters must be distributed before deployment. Also, the key escrow problem of IBE schemes is mitigated in WSN deployments as all nodes are typically owned by the same party. The work in [YWCR07] continues to propose an IBE-based broadcast encryption scheme to solve the problem of securely distributing the initial keys and parameters for μ-TESLA (see Section 4.2.1). Identity-based encryption for heterogeneous sensor networks is implemented in [SC09]. The authors propose to offload the larger computation load to the Cluster Head (CH) of the WSN and evaluate time and energy consumption for two common sensor node architectures. Recent work [FG10] also presents a lightweight id-based DH key agreement that does not use bilinear mappings and is as efficient as the best known authenticated DH key exchange protocols.

As no direct hardware support for ECC is currently available in sensor nodes, the authors of [YSF09] successfully evaluate the use of Digital Signal Processors (DSPs) for accelerated ECC computations on sensor nodes.

4.2.2 Secure Boot

Secure Boot describes a security mechanism which ensures that a platform only loads allowed components during bootstrap phase. If invalid code is detected the bootstrap process is interrupted and the code is not loaded [TY94].

AEGIS [AFS97] is a comprehensive architecture that applies this concept to PCs, so that the BIOS and all subsequently loaded components each verify the respective following component before delegating control to it. In case of AEGIS, the root of this chain is the immutable part of the BIOS that is loaded first to initialize the base system and load subsequent expansion ROM and the bootloader. The system was later extended with an automated recovery mechanism that allows the bootloader to retrieve an authenticated operating system over network in case the local system is compromised [AKFS98].

In the area of WSN, such mechanisms did not yet appear to find application. Due to the threat model and the requirement to reprogram nodes for re-deployment in other applications, it must be expected that the adversary is able to use any reprogramming facility that is not cryptographically protected. For performance reasons sensor systems also tend to be less complex and modular, limiting the benefit of the secure boot concept.

4.2.3 Secure Routing

While secure routing for homogeneous wireless sensor networks has been widely studied during recent years, only a few works address secure routing in Heterogeneous Wireless Sensor Networks (HSNs). In contrast, many secure routing protocols for homogeneous networks were proposed, but these cannot be applied directly to heterogeneous networks. However, they can be adopted for a single layer of the HSN hierarchy. Thus, in the following we survey existing secure routing protocols for both, heterogeneous and homogeneous networks.

Secure routing protocols for homogeneous sensor networks. One can differ two routing strategies for homogeneous sensor networks: Location-based and flat-based. Location-based routing protocols use information about node location to make forwarding decisions, while flat-based routing is used in networks without knowledge of location information.

Location-based secure routing protocols. The integration of trust-based reputation systems into geographic routing protocols like Greedy Perimeter Stateless Routing (GPSR) [KK00] was proposed in [TDBH04, HLK07, ZLV$^+$09]. Reputation systems associate a trust score with each sensor node in the network. They honor well-behaving nodes and punish suspicious behavior, providing means to identify malicious or selfish sensor nodes and to select well-behaving nodes for forwarding. However, these protocols do not assume the location information advertised by nodes to be verified, although trustworthy location information is essential to make correct forwarding decisions. Thus, a malicious node can falsify its location and compromise the basis of location-based routing.

Resilient Geographic Routing (RGR) [AGKL05, KLAG06] is also based on GPSR [KK00]. It employs a trust-based route selection based on locally generated trust information assigned to neighboring nodes and multipath forwarding. In contrast to [TDBH04, HLK07, ZLV$^+$09], RGR relies on verified location information certified by trusted nodes. Additionally, RGR employs rate control and packet scheduling to defend against data flooding.

Secure Implicit Geographic Forwarding (SIGF) [WFSH06] and its enhanced version Dynamic Window Secured Implicit Geographic Forwarding (DWSIGF) [HIJM09] are secure counterparts of Implicit Geographic Forwarding (IGF) [BHSS03], the stateless hybrid MAC/network routing protocol. SIGF comprises of three layers: SIGF-0, SIGF-1, and SIGF-2. Each layer provides different trade-offs between security and required resources. SIGF-0 is a lowest layer, it provides only probabilistic defenses against attacks, SIGF-1 integrates a trust-based reputation management system, while SIGF-2 enables cryptographic operations to provide authenticity, confidentiality, integrity, and freshness of messages. Additionally, DWSIGF provides an additional countermeasure against a protocol specific Clear-to-Send rushing attack that allows a malicious node to be selected for forwarding. In contrast to RGR, protocols from the SIGF family do not rely on certified location information. However, the trust-based management system implemented in the

SIGF-1 layer can mitigate location falsification attacks as it includes a metric to reflect location consistency reported by a node[6].

Secure routing protocols with a flat-based routing strategy. INtrusion-tolerant routing protocol for wireless SEnsor NetworkS (INSENS) [DHM06a] is a centralized link-state routing protocol, where routing information is first calculated in a centralized manner by a trusted base station based on the link and neighbor information gathered from the network and then securely distributed among the nodes. Security of route establishment phase of INSENS was formally proven in [ABV07]. While a design decision to assign route calculation to a trusted party implicitly eliminates many potential attacks, its centralized nature makes it unsuitable for TeSOS, because cluster heads (equivalents of a base station in the cluster) are untrusted in the TeSOS adversary model.

Secure SPIN (S-SPIN) [TL09] is a secure version of Sensor Protocols for Information via Negotiation (SPIN) [KHB02], the representative of data-centric routing protocols. In data-centric routing protocols, a sensor node interested in the particular data transmits a data request, describing the data it is interested in, to all nodes using flooding. The nodes possessing the data reply to the request. SPIN is much more efficient than classical flooding, as it avoids transmission of redundant data among sensors by negotiating individual sensor observations before transmitting the actual data. S-SPIN extends the basic SPIN with a Message Authentication Code (MAC) to ensure authenticity and integrity of SPIN messages. S-SPIN has been proven to be secure in the model proposed in [ABV06].

Secure-tinyLUNAR [Á09] is a secure version of Tiny Lightweight Underlay Ad hoc Routing protocol (tinyLUNAR) [Osi07] protocol, which is designed to support multiple communication paradigms at the same time: Data-centric, geographic-based and address-centric. Multi-addressing is a unique feature of tinyLUNAR which might be desired by certain application scenarios. However, functional universality of the protocol comes at a price of increased complexity. Moreover, tinyLUNAR supports reactive routing with on-demand route establishment, while the TeSOS network requires a proactive routing strategy which is more essential for monitoring applications.

Another protocol for networks with direct node-to-node communication proposes a recursive grouping algorithm to pattern establish routing tables and network addresses [PLGP06]. The deterministic nature of the grouping algorithm limits ability of compromised nodes to affect routing information. However, because the grouping algorithm assigns network addresses together with routing information, this limits its applicability for the intra-cluster communication, as network addresses of each node must be unique within the whole network, but not only within the cluster. Moreover, the communication pattern typical for the TeSOS network is node-to-sink rather than node-to-node.

Secure routing protocols for Hierarchical WSN. Two-Tier Secure Routing (TTSR) protocol [DGXC07] represents the network as two tiers, one tier is responsible for the inter-cluster communication, while another one for the intra-cluster communication. Location-based routing strategy is applied for communication across cluster heads, while tree-based routing is used for communication within the cluster. TTSR cannot be applied to TeSOS network, as it assumes weaker adversary model: Cluster heads are trusted in TTSR, while in TeSOS WSN the adversary is able to compromise them (with accordance to the TeSOS adversary model formulated in Section 3.1.3).

Resource Oriented Security Solution (ROSS) [CC07] is a framework that aims at protecting a network layer of HSNs from malicious attacks. The adversary model of ROSS assumes that cluster heads are not trusted, however, no proactive measures are taken against adversarial CHs, but the framework relies on the assumption that compromised nodes are somehow detected,

[6]It is natural to assume that initially any network has no compromised nodes. When a node gets compromised, it replaces true location information with a forged one, thus the reported location information changes.

and subsequently excluded from the network. This reactive strategy is not very suitable for the TeSOS network, as it is preferable to tolerate possible compromises of the cluster heads rather than to detect them.

4.2.4 Secure Wake-Up and Attestation

Secure wake-up can be divided into two problems, the security of sleep cycles of WSN nodes and the integrity of nodes after wake-up.

Secure Sleep Cycles Not much has been published on the security of sleep cycles in WSNs. As described in Section 4.1.2, denial of sleep attacks are very difficult to mitigate. It is typically insufficient to validate incoming service requests since the communication interface itself and message processing and validation already consume a significant amount of energy. One approach is to reduce wake-up cycles and the need for sensors to be woken up through communication signals. The authors of [LCCK07] present a detailed classification and analysis of events in a WSN to optimize this behavior. Another approach presented in [SBS02] uses a secondary low-power radio that wakes up the energy-intensive components after the wake-up command was validated. The authors of [FH09] adapt this idea for sensor networks and propose the use of authorization cookies using symmetric cryptography that are directly verified by the firmware of the low-power radio.

Endpoint Integrity The goal of endpoint integrity is to let a remote verifier have assurance in the integrity of a sensor platform at a given point in time, i.e., to exclude the possibility of malicious or unwanted software running on the device. This is a difficult problem as a compromised platform must be assumed to lie about its internal state towards a verifier. There are two known approaches to this problem: Either a special component of the system is assumed to be robust against compromise and can thus be used to report the state of the overall system, or the attestation procedure is designed such that it is hard for an attacker to emulate.

A well-known example for the former solution is the Trusted Platform Module (TPM) [TCG05b, TCG04] developed by the Trusted Computing Group (TCG). In this design, a hardware component is introduced that maintains a secure log of the system state. When receiving a challenge from the verifier, the hardware module creates a report from this system log that can be used by the verifier to evaluate if the remote platform can be trusted or if unknown code was loaded.

The latter solution, also known as software attestation [SPvDK04] exploits the fact that an adversary must emulate the attestation procedure to create a result that is accepted by the verifier. Assuming that the adversary is not able to physically modify the platform, [SPvDK04, SLS+05] thus implement the attestation procedure such that any emulation leads to delays that can be detected by the verifier. The method was adapted for secure software update for sensor sensor networks in [SLP+06]. Unfortunately, software attestation is very difficult to implement securely [SCT04, CFPS09] and requires impractical assumptions like a secure channel to the compromised platform.

Generally, while such measures make it harder for an adversary to remain undetected, there are also a number of unsolved problems: It is still unclear what properties must be measured and if specific properties exist at all whose state can be used to reliably detect modification of running software. In general, execution of code depends on the data to be processed. Implementation flaws can be used to give the data complete control over the program flow, even if it resides in not executable memory regions [BRSS08]. However, integrity measurement of the data that some software is working with is only meaningful if the verifier can validate this measurement, which makes it hard to validate unanticipated data like regular sensor measurements or user (adversary) input [CFPS09, SCT04].

4.2.5 Secure Time Synchronization

Secure Time Synchronization, the identification of clock offsets between peers in the presence of adversaries, has multiple known and practical solutions in area of network systems [SBK05]. In case of WSNs however, [MRS05, EE01, SBK05] argue that known fault-tolerant solutions for distributed systems like the Flood Time Synchronization Protocol (FSTP) [MKSL04] or the Network Time Protocol (NTP) [Mil91] are not suitable for use in WSNs. Due to the strong hardware constrains, even listening for radio broadcasts, which some of the existing protocols use for efficient information exchange, is considered highly expensive. Additionally, WSNs scenarios are complicated by stronger adversaries that can delay or even speed up the flow of packets. It is shown in [JDUX09] that secure distributed time synchronization in face of compromised nodes is NP-hard in general and still polynomially hard in case of fully connected network subgraphs, e.g., within a local WSN cluster.

As pointed out in [MRS05], the easiest way to address this problem is to not require time synchronization at all. If time synchronization is required by the application, the algorithm should be reviewed to identify the specific requirements. For instance, the μ-TESLA protocol presented in Section 4.2.1 does not require a very precise synchronization. On the other hand, a localization service based on triangulation[7] of sound measurements requires highly precise synchronization, but only for the corresponding local subgraph of the WSN [MRS05]. In both cases, the synchronized time must not necessarily be synchronized with an external clock, like the standard Coordinated Universal Time (UTC).

In correspondence with [JDUX09, Ya07], we categorize WSN time synchronization protocols as creating either *relative* or *absolute* synchronization amongst the participants. In case of absolute time synchronization, there is obviously always a reference that acts as sender to synchronize recipients. However, in case of relative time synchronization, we can further distinguish sender-to-receiver and receiver-to-receiver synchronization: In contrast to sender-to-receiver synchronization, which uses a sender as authority regarding the current time, receiver-to-receiver synchronization only establishes a common time reference between neighbors by identifying their clock offsets relative to a third party. Further, we can distinguish single-hop from multi-hop synchronization in that multi-hop protocols attempt to synchronize nodes that are unable to reach each other directly.

WSN-specific attacks on time synchronization protocols were first described in [MRS05]. Apart from illustrating attacks on predominant time synchronization methods for WSNs, the authors of [MRS05] also made first suggestions for secure time synchronization. In case of single-hop sender-receiver synchronization, they suggest to use μ-TESLA to authenticate messages of the sender. For receiver-receiver synchronization, they suggest to randomly elect a node to act as sender. In case of multi-hop synchronization, the single-hop synchronization protocol is used iteratively along the path to the target node. For additional security, the authors of [MRS05] propose seldom μ-TESLA-protected broadcasts of the current WSN time by central authorities to reduce the overall error margin a possible adversary may introduce. Similar to [SZC07] they further suggest to use redundant synchronization with different nodes to detect deviations introduced by adversaries and to discard synchronization messages from such nodes (containment).

The problem of timeliness of time synchronization messages was identified in [GvHS, GPvS08]. The works propose protocols for pair-wise, group-wise and multi-hop time synchronization, all of which leverage authenticated time stamps and the observation that the average sender-receiver delay for direct neighbors mostly depends on the speed of the radio, which allows to detect and discard delayed or speeded packets. Secure Pairwise Synchronisation (SPS) is the base protocol of the suite, using a simple challenge-response protocol to establish the clock offset to a neighbor.

[7]Triangulation is a technique to localize an object's position relative to a receiver based on the time that it takes for, e.g., radio emissions, to reach that receiver. It takes at least three receivers to unambiguously locate an object within a two-dimensional plane. When combining the relative measurements and receiver positions, the estimate of the position of the object is determined by the accuracy of the receiver's clock synchronization.

It is used in Secure Groupwise Synchronization (SGS) to synchronize each member of a group with every other member using a batched broadcasts. Secure Transitive Multi-hop Synchronization (STM) on the other hand is intended to establish a global time reference in a WSN using SPS in an iterated fashion. Identification of malicious nodes in SGS is possible but requires computation and communication effort that increases at least linearly with the group size. To prevent compromised nodes in STM from providing arbitrary time references to their children, [GPvS08] also proposes the use of multiple alternate paths.

TinyReSync [SNW06] is a synchronization protocol implemented for TinyOS [HSW+00b]. It uses SPS from [GvHS] but inserts time stamps at the MAC-layer as proposed in [GKS03]. After local (relative) synchronization, the global WSN time is established via iterated and repeated broadcast of the local clock differences by each node in the WSN. Additionally, TinyReSync authenticates timestamps and timeliness of broadcasts using the improved μ-TESLA described in Section 4.2.1. Since each node repeatedly broadcasts its time difference to the common WSN time source, a node that receives t time references can trivially identify up to $\frac{t-1}{2}$ compromised nodes.

Another approach to μ-TESLA authenticated time synchronization is presented in [YQF07], where initial synchronization is established using a global master key that is purged after initialization of the network. The authors of [San07] use SPS together with clock skew estimation from [GTS+09] to synchronize a hierarchical network in three phases, by synchronizing clusters, cluster heads, and optionally the resulting overall WSN time to an external reference.

4.2.6 Trust Management

Due to the distributed and highly exposed nature of WSN deployments, the trustworthiness of peer nodes is a central problem. While trust management generally also includes matters like key distribution and authorization, this section only reflects trust management as perceived in sensor network literature, which concentrates on identifying malicious or selfish sensor nodes to prevent them from influencing network operation.

To achieve these goals, trust management systems collect evidence of good or bad behavior, recorded either through direct observation or in form of second hand information from other nodes. Examples are recordings of similar sensor data reports from other nodes or confirmations of successfully routed data packets. Depending on the architecture, the evidence is evaluated on the individual nodes (distributed design) or gathered at central authorities. The first approach results in individual (asymmetric) trust relations, while the centralized design produces a globally consistent and complete view of all trust relations at the cost of additional communication. Depending on the kind of evidence collected and the purpose of the network, the assigned trust levels can be used to improve routing decisions, data aggregation, election of cluster heads or other critical operations. A more detailed discussion and classification of trust management and reputation systems can be found in [HZR09].

In one of the first works on trust management in sensor networks [KR05], the authors propose that the Cluster Head (CH) compares reported sensor measurements to those of physically close neighbors nodes. After statistical analysis to detect outliers, a Trust Index maintained by the CH is updated to reflect the reliability of the nodes based on good or faulty data reports. Nodes with high trust levels are preferred in future CH elections. Further, Shadow Cluster Heads are introduced as fall-back to prevent the loss of the Trust Index table. The process is modified in [CPG06, PC07] such that each node in a cluster maintains its own record of evidence for each neighbor. The resulting trust values are used only when electing the new cluster head, i.e., each node will elect the node with the highest trust level in its local trust table. The opposite approach is used in [SJL+06] to achieve higher scalability in case of distributed WSNs. In this design, a hierarchy of CHs are used to organize the WSN and each CH computes an aggregated trust value from its respective cluster members. Each cluster head reports the aggregated value to the next higher CH until the Base Station (BS) is reached. The BS then classifies the reported trust values and broadcasts which clusters are *trusted*, *untrusted* or *uncertain*.

Since these works are prone to bad-mouthing attacks (see Section 4.1.2), [CWZG07] introduces an agent-based trust management where dedicated authorized agent nodes are inserted in the WSN to collect evidence and report computed trust values. In contrast, the work in [CWZG07] proposes to use software agents that run in secure isolated execution environments that are assumed to be available on each sensor node. The agents monitor the actions of their local platform and provide certificates to local applications that can be used to indicate the trust level of the local platform towards peers.

Later systems like [GBS08, Zia08, TMK+09] make use of reputation models to improve on their trust inference mechanism. The concept of reputation allows a more accurate evaluation by including second hand evidence with the appropriate trust factor of the reporting party. Further, it allows to include multiple types of evidence in correspondence to their practical impact and to emphasize current behavior over older evidence (aging). As emphasized in [HZR09], the association of recorded evidence with the corresponding reporting identities is also essential to mitigate bad-mouthing and Sybil attacks.

Other works use second hand information to improve data aggregation [GBS08] and network routing [Zia08]. Both are prone to bad-mouthing attacks, however, [GBS08] mitigates this effect by prioritizing first hand evidence and considering the trust level of nodes that report the second hand evidence. In case of network routing decisions, sensor nodes with high trust can concentrate work load onto them and quickly exhaust their battery. To avoid this, works [TMK+09, SPT+09] thus also include the remaining battery power into the calculations. To collect first hand evidence on good routing decisions, [Zia08] proposes to let the sensor node listen after sending to confirm if the chosen neighbor correctly forwards the message. However, to fully prevent malicious manipulation it is required that the final receiver of a message acknowledges the received message [TMK+09]. Other types of evidence are the willingness of a node to participate in the propagation of second hand evidence and the proper use of cryptographic protocols, e.g., properly authenticated messages [TMK+09].

As shown in [JIB07, HZR09], many methods can be used for calculation and representation of trust. Instead of employing arbitrary metrics, [HZX08, LTS09] use the notion of entropy to represent the uncertainty of a node about the trustworthiness of another. An evaluation of the proposed trust management systems and a list of best practices can be found in [LRAFG10]. An analysis of the energy consumption of three reputation-based trust management systems is presented in [SLL09], however, no practical energy consumption measurements are provided.

4.2.7 Intrusion Detection

In recent research, there is a trend to design IDSs which are able to detect two and more attacks in a WSN rather than to defend against a specific attack. Two major models of intrusion detection include anomaly detection and misuse detection [KV02] (signature-based) detection. Anomaly detection builds a model of normal behavior, and compares the model with detected behavior. Such a system may be able to detect unknown attacks, but can also produce high rate false positive alarms. Signature-based systems compare observed behavior with known attack patterns (signatures). They have higher accuracy, but detect only known attacks and may require significant amount of memory to store all known signatures.

A complete architecture for IDSs in WSN typically has three building blocks [KDF07]: (i) network monitoring, (ii) decision making and (iii) action.

In a monitoring phase, network nodes perform detection by analyzing local information and overhearing neighbor communication. Most common approach to detect intrusion in WSNs is anomaly-based [dSMR+05, OM05, LNLP06, BG06, KDF07, HHC07, KGD08b, YT08, HHJ09, PGS09, BMAAdG09]. Some of them analyze statistics of traffic patterns [OM05, LNLP06, BMAAdG09], while others are rule-based, i.e., use rules which characterize normal behavior and monitor any behavior which breaks these rules. Systems based on misuse detection are not common in WSN due to the imposed overhead, but they exist, e.g., the IDS presented in [YWL09] utilizes a mixture of both, anomaly and misuse detection approaches.

In the decision making phase, the decision can be made either by a certain trusted sensor node (e.g., by a cluster head in [HHJ09]) or nodes can cooperate in order to produce a global intrusion detection decision [HHC07, KDF07, KGD08b, KBG+09], e.g., using voting schemes. When global decision is produced as positive, the network should perform the predefined action to mitigate or defend against the attack.

Possible defense actions can be divided into two subcategories [KDF07]: direct and indirect response. Direct response assumes excluding the suspicious node from any paths and forcing regeneration of new cryptographic keys with the rest of the neighbors. Indirect response involves notifying the base station about the intruder or reducing the quality estimation for the link to that node so that it will gradually loose its path reliability, or reducing trust value for that node in case if trust management system is applied.

In context of TeSOS project, we are interested at most in those IDSs which are designed for hierarchical heterogeneous WSNs. IDSs proposed in [LNLP06, HHC07, HHJ09, PGS09] are intended for clustered WSNs, but they assume re-election of the cluster heads and thus cannot be applied to heterogeneous TeSOS WSN. The only systems which assume a heterogeneous network structure are proposed in [SCKH05, YWL09]. The work in [SCKH05] concentrates on key establishment mechanisms and general architecture of the IDS without considering any concrete techniques to detect attacks (i.e., it is assumed that attack is somehow detected in a monitoring phase). In this approach, cluster heads are responsible for monitoring regular nodes in own cluster, while several elected nodes in a cluster perform monitoring of a cluster head. Work [YWL09] proposes the Hybrid Intrusion Detection System (HIDS) which utilizes rule-based anomaly detection to detect intrusion and then classifies the attack by applying neural network learning technique based on the Back Propagation Network (BPN) model. Simulation results demonstrate very high accuracy and low false positive rate. Unfortunately no information is provided about the imposed overhead, which can be high because of the two passes (first to detect attack and second to classify it) and misuse detection technique which is already concluded [KDF07] to be resource expensive.

5 Hardware for Embedded and Sensor Devices (M3)

The main goal of this chapter is to analyse and compare current microprocessor architectures concerning their applicability for sensor nodes in secure sensor networks. Therefore, the following sections define test criteria and analyse available hardware platforms with particular focus on:

- Applicability for using cryptographic algorithms in general
- Support of cryptographic algorithms in hardware
- Integration of Digital Signal Processors (DSPs)
- Applicability for sensor networks

This chapter is structured into four main parts. The first part (Section 5.1) defines test criteria to be used during the analysis and comparison of potential sensor network hardware. The second part (Section 5.2) analyses the suggested hardware, followed by the third part in Section 5.3, which compares the features of the analysed hardware architectures. The fourth part (Section 5.4) analyses the energy efficiency of each hardware architecture.

5.1 Test Criteria

This section defines the test criteria to be used during the analysis and comparison of potential sensor network hardware. We distinguish the following seven categories:

1. *Interfaces:* Interfaces to the IT environment or external hardware devices, such as sensors
2. *Memory:* The type and amount of memory
3. *Performance:* The performance of the Central Processing Unit (CPU) and memory
4. *Security Extensions:* The availability of CPU-internal or -external security extensions
5. *Power Management:* The availability of sleep states of the CPU or external devices
6. *Assurance:* Whether the platform is certified according to some security criteria
7. *Additional Criteria:* Additional hardware-specific extensions or functions

The following subsections discuss each category in more detail.

5.1.1 Interfaces (Input/Output, I/O)

Communication between sensor nodes is one of the most essential functionalities of Wireless Sensor Networks (WSNs). Moreover, sensor nodes have to be capable to access additional measurement hardware, in order to collect and to capture data of their environment. Therefore, a WSN node has to include interfaces to communicate with its environment. The following list defines the I/O interfaces to be considered in this analysis:

- Network
  - IEEE 802.15.4 (ZigBee)
  - IEEE 802.11 (Wireless Local Area Network (WLAN))
  - IEEE 802.3 (Ethernet)
  - Other interfaces
- Peripheral/Programming
  - Universal Asynchronous Receiver/Transmitter (UART)
  - Universal Serial Bus (USB)
  - Secure Digital Input Output (SDIO)
  - General Purpose Input/Output (GPIO)
  - Serial Peripheral Interface Bus (SPI)
  - Inter-Integrated Circuit (I^2C)

Regarding communication between sensor nodes, LAN, ZigBee, and WLAN are considered. Regarding communication between sensor nodes and additional measurement hardware, UART, USB, SDIO, GPIO, SPI, and I^2C are considered. In some cases, additional crypto-hardware is connected, using one of the above mentioned interfaces.

5.1.2 Memory

The available embedded devices differ from each other, regarding the amount and type of memory they provide. We mainly focus on the amount of memory, which the embedded device is capable to use and the type of memory, which is used. The memory type is especially important, because some devices need very fast memory, while some other devices need additional memory to store data for internal backups of measured sensordata. We thus examine:

- Available amount of memory
- Memory-technology (Static Random Access Memory (SRAM), Synchronous Dynamic Random Access Memory (SDRAM), Flash, others)
- Memory-management with regards to quality, including durability or Mean Time Between Failures (MTMF)

5.1.3 Performance

Typically, embedded or wireless sensor hardware is designed to save as much resources as possible, thus performance has a low priority. However, secure embedded operating systems that use strong encryption and computationally intensive hashing algorithms typically need hardware with a higher performance. Based on the classification of sensor nodes into Big Nodes (BNs) and Small Nodes (SNs) (see Section 3.1.2), the following list defines performance aspects considered in this analysis:

- Memory performance (access speed of volatile storage and caches)
- Processor speed for operations such as hashing, encryption, or decryption
- Availability of cryptographic co-processors
- Network performance
- Storage performance (speed of persistent storage such as Electrically Erasable Programmable Read-Only Memory (EEPROM))

5.1.4 Security Extensions

Very often, hardware-based security extensions provide a higher degree of security than security mechanisms only based on software. Moreover, hardware-based security extensions are often more efficient in terms of speed and resource consumption. Therefore, it is analysed, whether the hardware platforms provide hardware-based security extensions, such as:

Security domains / Isolation: Security domains are used to segment existing IT infrastructure into logical zones with a common trust level. A security domain could be an isolated subset of a network or hardware/software system combined with the computing resources attached to that subset. Isolation is provided through software or hardware configuration (e.g., Virtual Local Area Networks (VLANs)), internal firewalls, or virtualisation. The level of security results from implementation of the policies, processes, and security technology deployed within a domain, as well as the isolation boundary that defines the domain edges. The most important techniques that help to secure WSNs are listed below:

- Virtual address spaces
- TrustZone (see Section 5.2.1)
- Protection Rings
- Other realisations of security domains

Secure or authenticated boot: As described in Section 4.2.2, secure or authenticated boot gives the possibility to ensure that a platform only loads allowed components during bootstrapping. We focus on the the following techniques, to ensure the unchanged software state of a sensor node.

- *Texas-Instruments M-Shield:* M-Shield is a security extension [JA] designed by Texas Instruments to provide a high-security solution inside mobile platforms. Key benefits of M-Shield are on-chip cryptographic keys, secure execution environment, secure storage, secure chip-interconnects, Standard API to connect with TrustZone, Tampering detection, and high-performance hardware-based cryptographic accelerators.
- *TPM:* A hardware device, protected against manipulation and designated for opt-in usage, providing protected capabilities and shielded locations. The Trusted Platform Module (TPM) is a passive component and contains engines for random number generation, calculation of hash values and RSA key generation. A TPM generates and stores keys, signs or binds data to the platform and provides secure storage of measurement information of the platform's current state. The TPM is available in two versions: The new version 1.2 specified in [TCG05a] and the deprecated version 1.1b specified in [TCG02]

- *Qualcomm SecureMSM:* Originally designed to provide Digital Rights Management (DRM) inside multimedia-oriented devices, Qualcomm designed a security extension called SecureMSM [Qua]. As stated by Qualcomm, the key functionalities of SecureMSM are trusted execution of applications, fine granularity of permissions of executable content and API access control, trusted boot with integrity checking of the mobile device software, and a secure file system to ensure integrity of sensitive data.

Secure storage: Many security primitives (ciphers, pseudo-random numbers, authentication codes) require the availability of cryptographic keys. However, due to the large scale of sensor networks and the limited hardware capabilities of sensor nodes, standard methods of key storing, such as storing the key on a persistent memory, are not applicable. A secure storage has to provide confidentiality, integrity, and freshness. In addition to [WRLZ09], describing techniques of a distributed data storage by proposing a novel dependable and secure data storage scheme with dynamic integrity assurance, a secure storage can be realised in hardware. Examples of security extensions that can be used to realise secure storage are M-Shield [JA], a TPM, or Qualcomms SecureMSM [Qua].

Random Number Generator: The "randomness" necessary to generate secure cryptographic keys requires a high quality, i.e. a lot of entropy, which cannot be generated by software mechanisms only. Therefore, a physical device designed to generate such a sequence of random numbers or bitstrings is required. The key material should be obtained from a true random source with high entropy so that the security relevant mechanisms inside a WSNs can trust on, e.g., secure cryptographic keys. In general, good mechanisms to generate secure randomness is a hardware-generator based on microscopic phenomena, such as thermal noise or the photoelectric effect, which are, in theory, completely unpredictable.

5.1.5 Power Management

As energy is the most relevant limiting factor for the operration of WSNs, especially in the context of small sensor networks, this section analyses which power management features the different hardware platforms support. It is also analysed, how efficient the hardware platform is, i.e., how much energy it requires for important operations, such as cryptographic operations (e.g., Advanced Encryption Standard (AES), Elliptic Curve Cryptography (ECC) or Secure Hash Algorithm-1 (SHA1)). In detail, the following aspects will be considered:

Types of sleep-modes: Some architectures already include different types of sleep modes within their own power management mechanisms. These sleep-modes are listed and described shortly.

Power-consumption: Depending on the processor speed, there is a strong interrelation between performance and consumed energy. Moreover, as the processor is one of the main energy-consuming parts of a sensor node, we focus on the consumption of an activated or deactivated radio. Since some architectures provide different types of sleep-modes, these sleep-modes are considered, regarding their reduction of the consumed energy.

Efficiency: Depending on the instruction set, the design, and additional power-management functionalities, processors consume different amounts of energy to realise the same software function. Especially cryptographic operation, such as AES, ECC, or SHA1, and other typical functions inside WSNs, strongly depend on efficient implementations inside software and/or hardware.

5.1.6 Assurance

To allow a high-level estimation of the quality of security mechanisms used inside WSNs, existing certificates and their availability inside WSNs are considered. A special focus will be on the following standards:

FIPS: Used as standards and guidelines, the United States federal government publicly announce their Federal Information Processing Standards (FIPS) [oSN] for use by all non-military government agencies and by government contractors. Especially the FIPS 140-standard with its four levels of certification are relevant in the area of WSNs since, they also focus on requirements for physical tamper-evidence and -resistance.

Common Criteria: The Common Criteria (CC) permits comparability between the results of independent security evaluations. The CC does so by providing a common set of requirements for the security functionality of IT products and for assurance measures applied to these IT products during a security evaluation. These evaluations are partly applied in WSNs and their used hardware platforms.

5.1.7 Additional Criteria

These criteria include organisational or even political aspects, such as the country where the embedded hardware is developed and produced or the price, which can also be important aspects to be considered.

Integration of signal processors: In some cases, available devices have integrated DSPs, which can be used for accelerated computation inside the sensor node.

Word size width of processor architecture: Depending on the available word size of the processor architecture, its Instruction Set Architectures (ISA) can be more or less powerful, i.e., operate on more data at the same time. Also, addressing high amounts of memory depends on the size of the used registers. For example, loading data from high memory addresses requires two or more CPU-cycles if the full address is bigger then the CPU's word size.

Manufacturer: In security related areas, such as military or governmental usage, it can be of interest who the manufacturer of the sensor hardware is.

Price: In WSNs with a huge amount of sensor nodes, the price of each node is also relevant, thus this criterion is also considered in this document. Additional costs not examined in this work are programming, maintenance, and deployment of the WSN.

5.2 Hardware Architectures

This section analyses and compares common hardware platforms used for embedded systems and sensor nodes. The analysed hardware architectures are:

- ARM-Architecture, especially Cortex M0 [ARMb]
- Leon2-Architecture [Gaic]
- Leon3-Architecture [Gaif]
- Atmel AVR XMEGA [Atma]

- Atmel SecureAVR [Atmf]
- Texas Instruments MSP430 [Texb]
- Trusted Sensor Node (TSN), based on Leon2 and provided by the Bundesamt für Sicherheit in der Informationstechnik (BSI)

The following sections discuss these architectures in detail, focusing on their specific features and differences. Section 5.3 summarises this chapter by a tabular comparison of these platforms, based on the criteria defined in Section 5.1.

5.2.1 ARM-Architecture

This section gives a short overview of the currently available ARM-Architectures followed by a more detailed look into the Cortex-M0 processor. Especially the Cortex-M0 processor has been designed by ARM Holdings (ARM) to provide a secure mobile computing device where less performance is needed.

Architecture Overview

ARM provides different sub-types of their hardware architecture. Figure 5.1 gives an overview of the currently available ones.

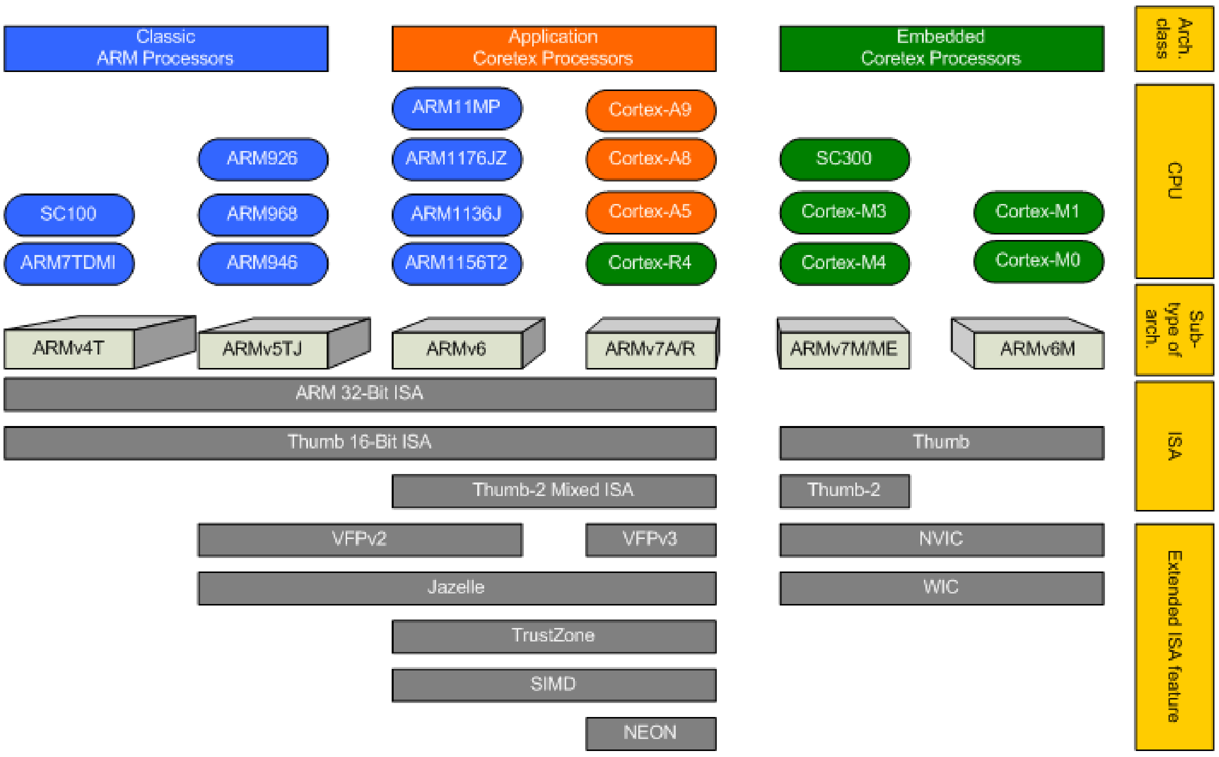


Figure 5.1: Different types of currently available ARM-Architectures

Each hardware architecture represents a family of ARM processors with its own set of features, such as Thumb, Thumb-2, Single instruction multiple data (SIMD), TrustZone, or Jazelle. Since this document focuses on hardware platforms that can be used as sensor nodes, we concentrate on the ARM Cortex-M series, which is designed for fast interrupt processing and cost-sensitive devices requiring high energy efficiency.

ARM TrustZone

Since security domains are a desired security feature, this section briefly introduce the ARM TrustZone technology.

As already described in Section 5.1.4, security domain structuring is an approach to segment existing infrastructure into logical zones, based on a common trust level. The ARM TrustZone extension provides hardware support for two separate address spaces, such that code executing in the non-secure world cannot gain access to an address space marked as secure. A special monitor mode secures transition between the two worlds (see Figure 5.2).

The TrustZone technology provides a secure environment for system features, such as key management and authentication mechanisms enabled by the operating system. The protection provided by the technology is useful for consumer privacy and extending a range of services, such as mobile banking and multimedia entertainment, to widespread consumer adoption and use.

Currently, TrustZone is implemented only in the ARM Cortex-A8, Cortex-A9, Cortex-A9 MPCore and ARM1176JZ(F)-S, thus it is not available in the Cortex-M-series.

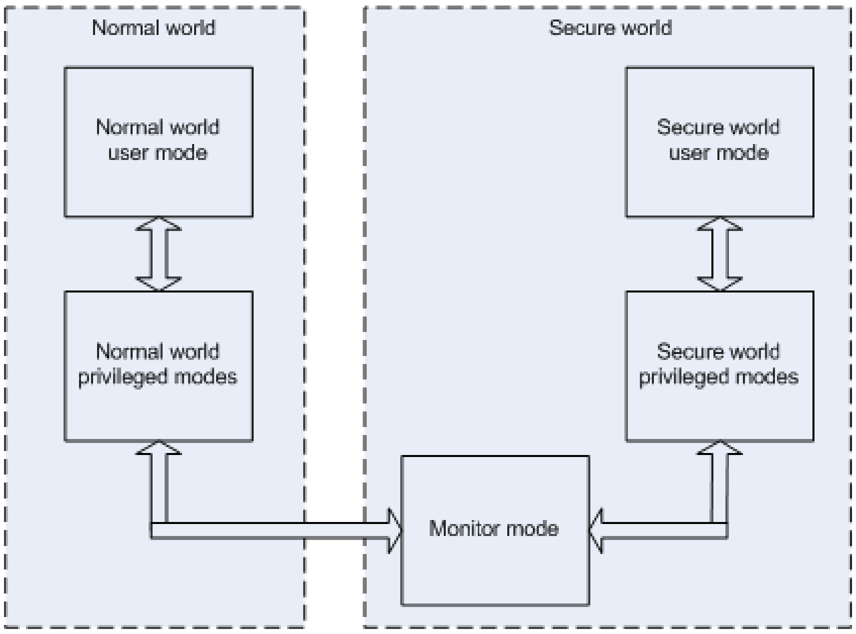


Figure 5.2: Modes in an ARM core using TrustZone

As Figure 5.2 illustrates, the physical processor core(s) that implement TrustZone provide two virtual cores, one considered non-secure and the other secure, and a mechanism to perform secure context switches between them, known as monitor mode. The value of the “NS bit”, a Secure Configuration Register (SCR) [ARMg] that can only be accessed in secure (privileged) modes, sent on the main system bus, is the identity of the virtual core that performed the instruction or data access. This enables trivial integration of the virtual processors into the system security mechanism; the non-secure virtual processor can only access non-secure system resources, but the secure virtual processor can see all resources. For additional information about ARM TrustZone we refer to [ARMi] and [ARMf].

Although it is assumed in Section 3.1.2 that SNs are not equipped with additional features, such as TrustZone, the ability to separate the physical processor core into two different parts could be used in BNs as an additional hardware security feature.

Cortex-M-series

Since the Cortex-M-series seems to be best suitable to be used as a basis for WSN nodes, especially for SNs, the following sections discuss this CPU family in more detail. Inside the Cortex-M-series, ARM implemented a couple of techniques that are shortly described in the following.

Thumb technology: Since memory in WSN nodes is very limited, all operations on these platforms have to be optimised for code size. The ARM Thumb technology [ARMe] is an extension of the 32-bit ARM-Architecture and a possible solution to solve this code size issue. The Thumb instruction set features a subset of the most commonly used 32-bit ARM instructions compressed into 16-bit wide opcodes. On execution, these 16-bit instructions are decompressed transparently in the instruction pipeline to full 32-bit ARM instructions in real time. Designers can use both 16-bit Thumb and 32-bit ARM instructions sets and therefore have the flexibility to emphasise performance or code size on a sub-routine level as their applications require.

Thumb-2 technology: In addition to the reduced code size realised by the Thumb technology, Thumb-2 [ARMc] reduces the code-size even more. Thumb-2 core technology adds a mixed mode capability to the CPU, defining an additional set of 32-bit instructions that execute alongside traditional 16-bit instructions in Thumb state. This reduces or removes the need for balancing ARM and Thumb code in a system, since the 32-bit Thumb-2 instructions do not need to be decompressed. Thumb-2 technology is a superset of Thumb technology and is backwards compatible with existing ARM and Thumb solutions.

NVIC: The Nested Vectored Interrupt Controller (NVIC) ([ARMd], Chapter 5, page 47) is an integral part of Cortex-M processors and provides the processors' interrupt handling abilities. The Cortex-M processor uses a vector table that includes the address of the function to be executed for a particular interrupt handler. On accepting an interrupt, the processor fetches the address from the vector table. To reduce gate count and enhance system flexibility, the Cortex-M processor uses a stack based exception model. When an exception takes place, critical general purpose registers are pushed on to the stack. Once the stacking and instruction fetch are completed, the interrupt service routine or fault handler is executed, followed by the automatic restoration of the registers to enable the interrupted program to resume normal execution. This approach removes the need to write assembler wrappers that are required to perform stack manipulation for traditional interrupt service routines based on the programming language C, which reduces code size and thus saves memory in WSNs. The NVIC supports nesting (stacking) of interrupts, allowing an interrupt to be serviced earlier by exerting higher priority.

WIC: The Wake-up Interrupt Controller (WIC) provides a low power interrupt detection logic that can emulate the full NVIC behaviour when correctly primed by the full NVIC on entry to very-deep-sleep. For low power applications, it is desirable to reduce the dynamic and static power-consumption of the processor while in deeper sleep modes. This can be achieved by stopping clocks, removing power from the processor or both. When powered off, the NVIC is unable to detect interrupts, so that knowing when to come out of sleep becomes problematic.

Unlike the NVIC, the WIC has no prioritisation logic. It implements a simple interrupt masking system, signalling for wake-up as soon as a non-masked interrupt is detected. The WIC is invisible to end users of the device.

Cortex M0 processor

For the purpose of this study, we focus on the ARM Cortex-M0 processor with its low gate count, an energy efficient processor based on the ARMv6M architecture that is intended for microcontroller and highly embedded applications. The Cortex-M0 is a configurable, 32-bit Reduced Instruction Set Computer (RISC) processor, has an AMBA High-performance Bus (AHB)-Lite interface and includes an NVIC component. It also has optional hardware debug functionality and can execute Thumb code.

Figure 5.3 illustrates a block diagram of the architecture overview of the Cortex-M0 processor.

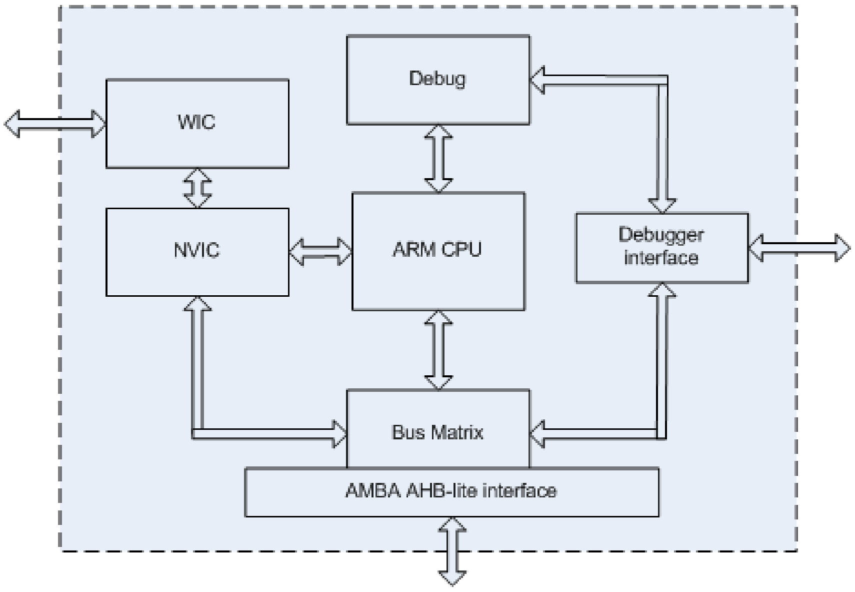


Figure 5.3: ARM Cortex-M0 processor design

Key features of this processor include Thumb instruction set with decreased code-size and power-control using NVIC and WIC, in order to permit a slower processor clock or increased time in sleep mode. Moreover, a 32-bit hardware multiplier is included in this architecture. The ARM Cortex-M0 can be configured with additional features, such as a multiplier optimised for speed or size, up to 32 external interrupts, up to four breakpoint comparators or up to two watchpoint comparators. The 32x32-bit multiplier of the processor can be implemented either as a fast single cycle array or a 32-cycle interactive multiplier, thus the processor can be optimised for power consumption or performance.

Power Management and Performance: For the Cortex-M0 processor, ARM estimates a power consumption of about 85 microwatts/MHz (0.085 milliwatts). Dhrystone benchmark [Ala] reaches up to 0.9 DMIPS/MHz [ARMh].

Applicability in WSNs: A manufacture that is currently using the ARM Cortex-M0 as a sensor node is NXP Semiconductors (NXP) [NXPb] with its LPC11xx microcontroller. NXP designed the device to be used in battery-powered systems, e-Metering, consumer peripherals, remote sensors or any 16/32-bit applications [NXPa]. Compared to other architectures, such as the TI MSP430 (see Section 5.2.7), the Cortex-M0 is not widely used in the area of WSNs.

5.2.2 Leon2-Architecture

The Leon2 is a Very-high-speed integrated circuits Hardware Description Language (VHDL)-model designed by Gaisler Research [Gaid] and is published as Open-Source component, based on the GNU Lesser General Public Licence (LGPL) license [Freb]. The Leon2 is a 32-bit RISC-processor build from the SPARC-V8-architecture, which is described in IEEE 1754 [IEEc]. An advantage of the LGPL license is that the Leon2 can be used as a core in a system-on-chip design without having to publish the source code of possibly additional used cores.

Architecture Overview

The Leon2 offers separate instruction and data caches, a hardware multiplier and divider, an interrupt controller, a debug support unit with trace buffer, two 24-bit timers, two UARTs, a power-down function, a watchdog, a 16-bit I/O port, a memory controller, an Ethernet Media Access Control (MAC), and a Peripheral Component Interconnect (PCI) interface.

New modules can easily be added, using the on-chip Advanced Microcontroller Bus Architecture (AMBA) AHB/Advanced Peripheral Bus (APB) buses (more details can be found in [ARMa]). The VHDL model is fully synthesisable with most synthesis tools and can be implemented on both, Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). Simulation can be done with all VHDL-87 [IEEa] compliant simulators.

Moreover, the Leon2 offers advanced fault-tolerance features to withstand arbitrary Single Event Upset (SEU) errors without loss of data. SEU-errors are changes of states caused by ions or electro-magnetic radiation striking a sensitive node in a micro-electronic device, such as in a microprocessor, semiconductor memory, or power transistors. The fault-tolerance is provided at design VHDL level, and does neither require an SEU-tolerant semiconductor process nor a custom cell library or special back-end tools. A block diagram of Leon2 is shown in Figure 5.4.

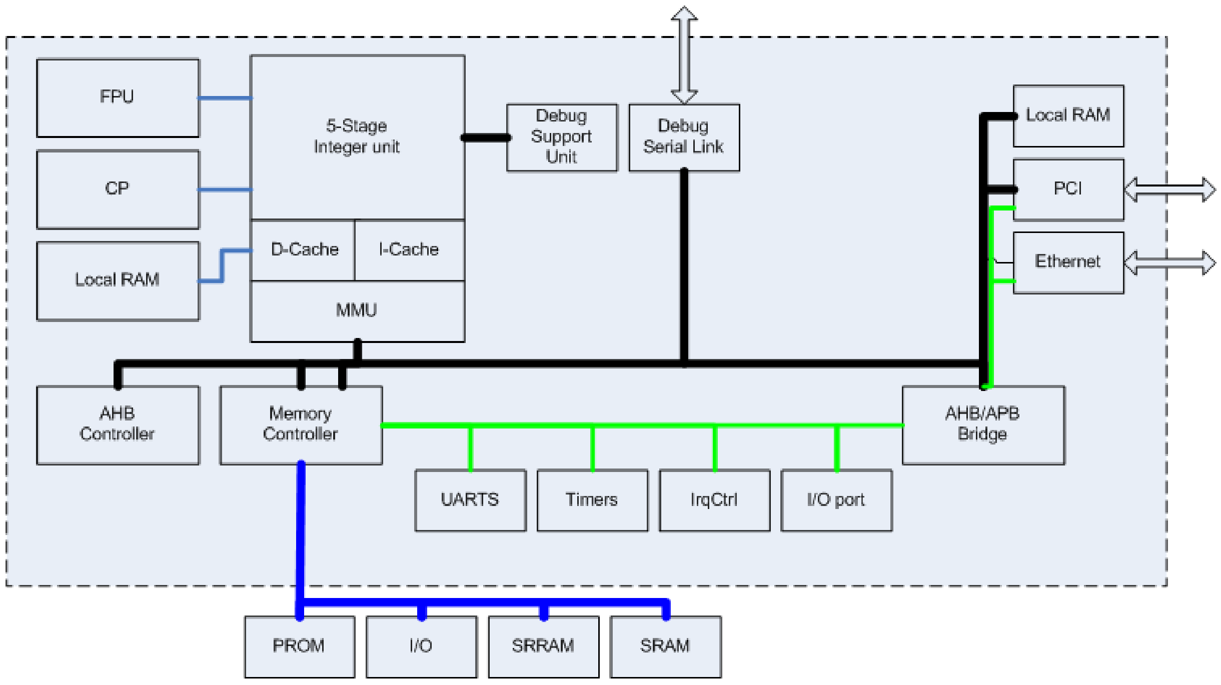


Figure 5.4: Leon2 processor design

Since the Leon2 is an open architecture, a comprehensive public documentation exists, which makes it possible to describe every element of the Leon2-Architecture in detail.

Integer unit: The Leon integer unit implements the full Scalable Processor Architecture (SPARC) V8 standard, including all multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC standard (2 - 32), with a default setting of 8.

Floating-point unit and co-processor: The Leon processor model provides an interface to the high-performance GRFPU (Gaisler Research floating-Point Unit (FPU)) [Gaia], or the Meiko FPU core (Sun Microsystems). Furthermore, a generic co-processor interface is provided to allow interfacing of custom co-processors. It is thus possible to attach additional modules, e.g., cryptographic co-processors, to the processor, which can be useful in the context of secure WSNs.

Memory management unit: The (optional) Memory Management Unit (MMU) implements a SPARC V8 reference MMU and allows usage of commodity operating systems, such as Linux or Solaris. The MMU can have a separate instruction buffer, data buffer, or a common Translation Look-aside Buffer (TLB). The TLB is configurable for 2 to 32 fully associative entries.

Debug support unit: The (optional) Debug Support Unit (DSU) allows non-intrusive debugging on target hardware. The DSU allows to insert breakpoints, watchpoints, and access to all on-chip registers from a remote debugger. A trace buffer is provided to trace the executed instruction flow and/ or AHB bus traffic. The DSU has no impact on performance and has low area complexity. Communication to an outside debugger (e.g., gdb [Frea]) is achieved by using a dedicated UART (RS232) or through any AHB master (e.g., PCI).

Memory interface: The flexible memory interface provides a direct interface Programmable Read-Only Memory (PROM), memory mapped I/O devices, SRAM and SDRAM. The memory areas can be programmed to either 8-, 16-, or 32-bit data width.

Timers: Two 24-bit timers are provided on-chip.

Watchdog: A 24-bit watchdog is provided on-chip. The watchdog is clocked by the timer prescaler. When the watchdog reaches zero, an output signal is asserted, which can be used to generate a system reset.

UARTs: Two 8-bit UARTs are provided on-chip.

Interrupt controller: The interrupt controller manages a total of 15 interrupts, originating from internal and external sources. Each interrupt can be programmed to one of two priority levels. A chained, secondary controller for up to 32 additional interrupts is also available.

Parallel I/O port: A 16/32-bit parallel I/O port is provided.

AMBA on-chip buses: The processor has a full implementation of AMBA AHB and APB on-chip buses. A flexible configuration scheme makes it simple to add new cores. Also, all provided peripheral units implement the AMBA AHB/APB interface making it easy to add more of them, or reuse them on other processors using AMBA. More information about the AMBA can be found in [ARMa].

PCI interface: A low complexity PCI interface can be enabled and can also be used for debugging.

Ethernet MAC: An Ethernet 10/100 Mbit MAC can be enabled. The MAC is based on the Ethernet MAC core from OpenCores ([Ope]), with an additional AHB interface.

On-chip RAM: A (small) on-chip Random Access Memory (RAM) can be attached to the AHB bus, providing fast local memory. The size can be configured from 1 to 64 kB.

Also, the Leon2-Architecture provides the possibility to integrate a hardware multiplier into the design. It was designed under contract from the European Space Agency [Eur], and is now available as a radiation-hardened components from Atmel (AT697 and AT7913). More information on the Leon2 can be found in [Gaie] or [Gaic].

Power Management and Performance

Since the Leon2 design does not include power management, it has to be implemented by the device vendor or other designers.

Using 4K + 4K caches and a 16x16 multiplier, the Dhrystone 2.1 benchmark ([Ala]) reports 1,550 iteration/s/MHz using the gcc-2.95.2 compiler. This translates to 0.9 Dhrystone Million Instructions Per Second (MIPS)/ MHz using the VAX 11/780 value a reference for one MIPS [Res].

Leon2 offers a 5-steps pipeline to execute commands, which is separated into the following:

- *Fetch:* Fetch the instruction from the memory, using the memory controller
- *Decode:* Decode the instruction and get the operands out of the register-window
- *Execute:* Operation is executed inside the ALU
- *Memory:* The Result from the execute step is stored in the data-cache
- *Write:* The Result from the execute step is written back to the corresponding register-windows

Assuming the necessary data is already in cache, almost all operations can be done in only one clock-cycle. Excluded are jump, load, and store operations. A complete table of cycles of important functions is illustrated in Section Table 5.1.

Table 5.1: Instruction timings inside Leon2

Instruction	**Cycles**
JMPL	2
Double load	2
Single store	2
Double store	3
SMUL/UMUL	1/2/4/5/35 (depends on multiplier conf.)
SDIV/UDIV	35
Taken Trap	4
Atomic load/store	3
All other instructions	1

More detailed information on power consumption of this architecture can be found in Section 5.4.2.

Applicability in WSNs

The Leon2 brings a lot of benefits for designing a sensor node. It is available for free, and designers can modify it to their own needs. An example for the suitability of Leon2 for sensor nodes in WSNs is the Trusted Sensor Node (Section 5.2.6).

5.2.3 Leon3-Architecture

As an advancement of the Leon2 architecture described in Section 5.2.2, the Leon3 is a 32-bit processor based on the SPARC V8 architecture with support for multiprocessing configurations. The processor is fully synthesisable and up to 16 CPU cores can be implemented in Asymmetric Multiprocessing (AMP) or Synchronous Multiprocessing (SMP) configurations. The multiprocessor core is available in full source code under GNU General Public Licence (GPL) for evaluation, research, and educational purposes. A low cost license is available for commercial applications.

Architecture Overview

The Leon3 provides hardware support for cache coherency, processor enumeration, and interrupts steering. A debug interface allows non-intrusive hardware debugging of both single- and multiple-processor systems, and provides access to all on-chip registers and memory. Trace buffers for both instructions and AMBA bus traffic are available. Each core can be configured to use an IEEE-754 [IEEb] compliant FPU for floating-point operations (for area critical designs one FPU can be shared between CPU cores). A SPARC reference MMU is provided for advanced memory management and protection. Figure 5.5 illustrates the architecture of a Leon3 processor.

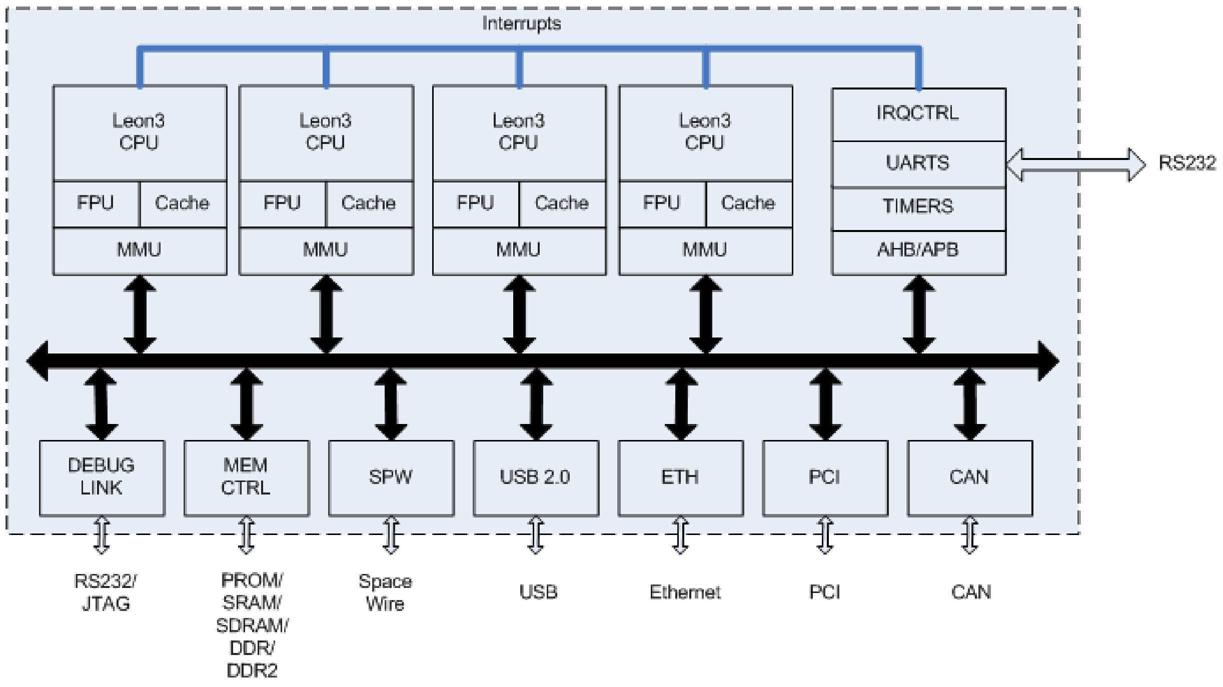


Figure 5.5: Leon3 Architecture Design

The Leon3 processor is highly configurable. The configuration of each processor in terms of cache size, FPU, and MMU usage can be individually defined. Asymmetric configurations, such as two main processors with FPU and MMU and two I/O processors, are supported. The Leon3 processor system takes full advantage of the plug&play capabilities of the GRLIB IP-library [Aer], increasing flexibility and reducing development time.

For SMP configurations, the operating systems Linux 2.6 SMP and eCos have been ported for Leon3. Linux 2.6 SMP is able to automatically load balance applications across multiple Leon3 cores, providing hardware/software architectures for high performance systems. For loosely coupled (message passing) AMP configurations, operating systems, such as RTEMS [Corc] and uCLinux [DI], are available. More information about operating systems can be found in Chapter 7.

A flexible implementation using between 1 and 16 processors, and sizing of both, data and instruction cache is between 0k and 2 MB across each CPU is possible. The GRLIB IP-library with plug&play functionality enables rapid and flexible SoC designs. Also, an optional IEEE-754 Floating Point Unit [IEEb] and a SPARC reference MMU can be added to the processor. Additional information on Leon3 can be found in [Gaig] or [Gaif].

Power Management and Performance

Individual processors can be shut down, which provides significant savings for dynamic power consumption. Also, each processor's clock CPU can be individually gated off in power down mode for further reduction of both, dynamic and static power consumption.

A typical configuration with four processors is capable of delivering up to 1600 Dhrystone [Ala] MIPS of performance.

With its SPARC V8 architecture multiprocessor-capable instruction set architecture used, 400 MHz on a 0.13 um ASIC process, giving up to 6400 Dhrystone MIPS of performance is reached. This brings up to 1.4 DMIPS/MHz, 1.8 CoreMark/MHz (using gcc version 4.1.2).

Applicability in WSNs

Leon3 is the successor of the Leon2. The Leon3 adds multicore options to the architecture and offers increased performance in combination with power management. Regarding WSNs, the Leon3 is less useful for SNs, but much more for BNs. The Leon3 offers a good starting point to design customised sensor device, since the design is using a public license and gives the designer the possibility to design a fast device in a relatively small amount of time. Currently, there are no sensor nodes publicly available, which use the Leon3-Architecture.

A special version of Leon3, the LEON3FT-RTAX Fault-tolerant Processor is used in combination with the radiation resistant FPGA made by Actel in the satellite systems Chandrayaan-1, Prisma, and the Taiwanese Argo. This shows the potential of Leon3 to be used in a very low power area with suitable performance and fault tolerance even over years [Gaih].

5.2.4 Atmel AVR XMEGA

In the year 2008, Atmel announced their XMEGA-Controller, which is based on their 8-bit AVR core with additional features in peripherals and functions. The following section introduces the architecture, followed by aspects of power and memory management.

Architecture Overview

The XMEGA is a family of low power, high performance and peripheral rich Complementary Metal-Oxide Semiconductor (CMOS) 8/16-bit microcontrollers, based on the AVR enhanced RISC architecture.

The AVR CPU combines an instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction, executed in one clock cycle. The resulting architecture is more code efficient and achieves many times faster throughput than conventional single-accumulator or Complex Instruction Set Computer (CISC) based microcontrollers. Also AES and Digital Encryption Standard (DES) are supported in hardware. The block diagram in Figure 5.6 introduces the architecture of the Atmel XMEGA.

The XMEGA devices provide the following In-System Programmable Flash with Read-While-Write capabilities, Internal EEPROM and SRAM, four-channel Direct Memory Access (DMA) controller, eight-channel Event System and Programmable Multi-level Interrupt Controller, up to 78 general purpose I/O lines and a 16- or 32-bit Real-Time Clock (RTC). Furthermore, up to eight flexible 16-bit Timer/Counters with compare modes and power management, up to eight UARTs, up to four I^{2}C and System Management Bus (SMB) compatible Two Wire Interfaces (TWIs) and a lot of other features are offered by the AVR XMEGA. Interesting for the purpose of TeSOS is the availability of an AES and DES cryptographic engine in hardware.

The program Flash memory can be reprogrammed in-system through Joint Test Action Group (JTAG) interface. A bootloader can use any interface to download the application program to the flash memory. The bootloader software in the boot flash section will continue to run while the application flash section is updated, providing true read-while-write operation. More details on the XMEGA architecture can be found in [Atmd] and [Atmb].

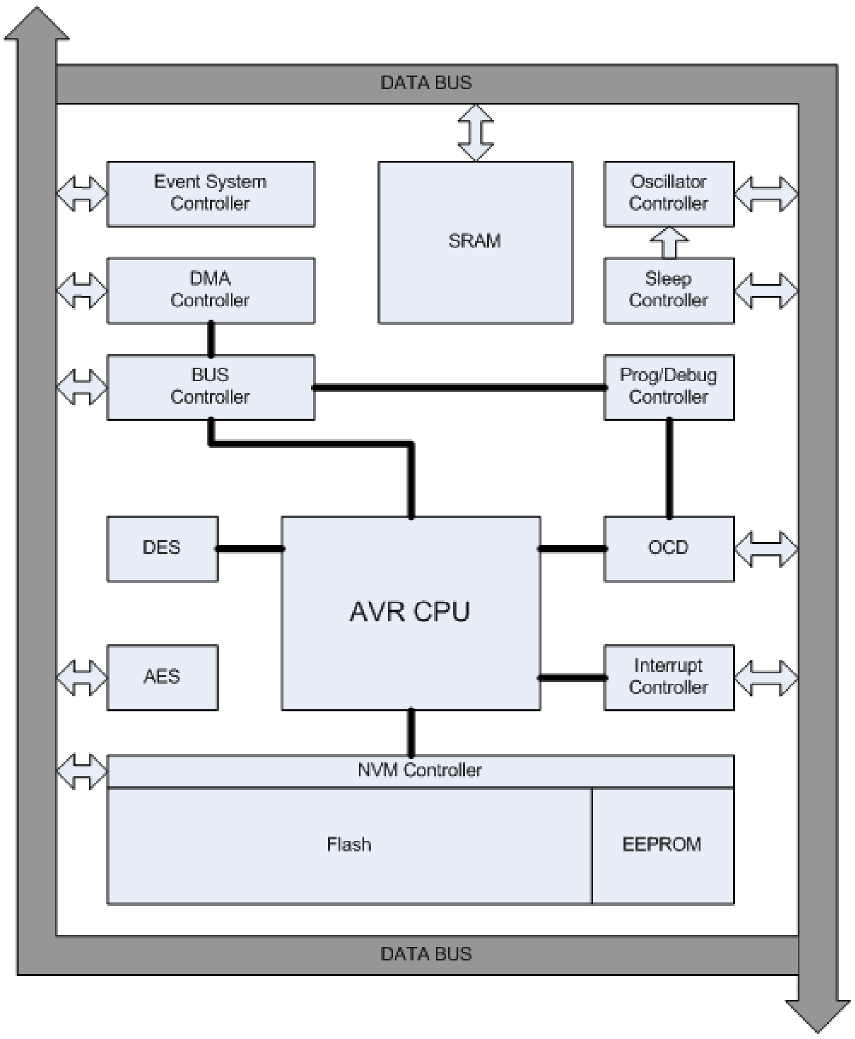


Figure 5.6: XMEGA block diagram

Power Management and Performance

The XMEGA devices have five software selectable power saving modes.

- *Idle*: The Idle mode stops the CPU, while allowing the SRAM, DMA controller, event system, interrupt controller, and all peripherals to continue functioning.
- *Power-down*: The Power-down mode saves the SRAM and register contents but stops the oscillators, disabling all other functions until the next TWI or pin-change interrupt, or Reset.
- *Power-save*: In Power-save mode, the asynchronous RTC continues to run, allowing the application to maintain a timer base while the rest of the device is sleeping.
- *Standby*: In Standby mode, the Crystal/Resonator Oscillator is kept running while the rest of the device is sleeping. This allows fast start-up combined with low power consumption.
- *Extended Standby*: In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

To further reduce the power consumption, Atmel included so-called Power Reduction Registers. These registers provide a method to stop the clock of individual peripherals. When this is done, the current state of the peripheral is frozen and the associated I/O registers cannot be read or written. Resources used by the peripheral will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock. Enabling the clock to a peripheral again puts the peripheral in the same state as before it was stopped. This can be used in idle mode and active mode to reduce the overall power consumption significantly. In all other sleep modes, the peripheral clock is already stopped. Not all devices have all the peripherals associated with a bit in the power reduction registers.

With XMEGA, using second generation picoPower [Cora], battery life is further increased by additional features like true 1.6V operation and a combined watchdog timer- and Brown-Out-Detector (a detector for a drop in voltage in an electrical power supply) current consumption of only 1 uA. True 1.6V operation means all functions, including Flash reprogramming, EEPROM write, analogue conversions, and internal oscillators, are operative at 1.6V. In battery powered applications, such as mobile phones, AVR XMEGA devices can be connected to a 1.8V (+/- 10%) regulated power supply to save cost and increase battery life. In Power-down mode with RAM retention, current consumption is 100 nA. The RTC function using a 32 kHz crystal oscillator has a power consumption of only 650 nA [Atme].

Atmel also offers a document on how to minimise the power consumption of the XMEGA, including C code, which makes it possible to decrease development time of a low power device [Atmc].

By executing instructions in a single clock cycle, the XMEGA achieves throughputs of up to 1 MIPS per MHz. The clock of the internally used AVR CPU can be up to 32 MHz.

Applicability in WSNs

The XMEGA devices are general purpose microcontrollers well suited for a variety of applications, including audio systems, ZigBee or medical applications, power tools, board controllers, networking, metering, optical transceivers, or motor control. Including its AES and DES cryptographic engines in hardware, it offers a wide spectrum of functionalities for WSNs and may be a good choice for TeSOS BNs.

5.2.5 Atmel SecureAVR

Originally Atmel has different kinds of product families with its own sub-families. One of those families is the Atmel XMEGA (see Section 5.2.4), which is designed to be used as a sensor platform. Another platform is the Atmel SecureAVR family, which contrary to the XMEGA, is designed to run inside smartcards. However, we discuss this platform here, since the functionalities protect some of the assets described in Section 3.1.1, e.g., parts of the sensor code or key information can be securely stored inside the device.

Architecture Overview

The AT90SC family Secure AVR integrates a Random Number Generator (RNG), cryptographic co-processor, and on-chip security features to fulfil assurance requirements of EAL5+ [Cri]. The block diagram in Figure 5.7 gives an overview of the architecture.

The device performs encryption functions in real time, enabling Global System for Mobile communications (GSM) Subscriber Identity Module (SIM) cards, Internet transactions, pay TV, or banking designs. To also increase the performance of RSA, Digital Signature Algorithm (DSA), ECC, and Diffie-Hellman, it optionally supports a cryptographic co-processor called AdvX, an n-bit multiplier-accumulator dedicated to perform fast encryption and authentication functions.

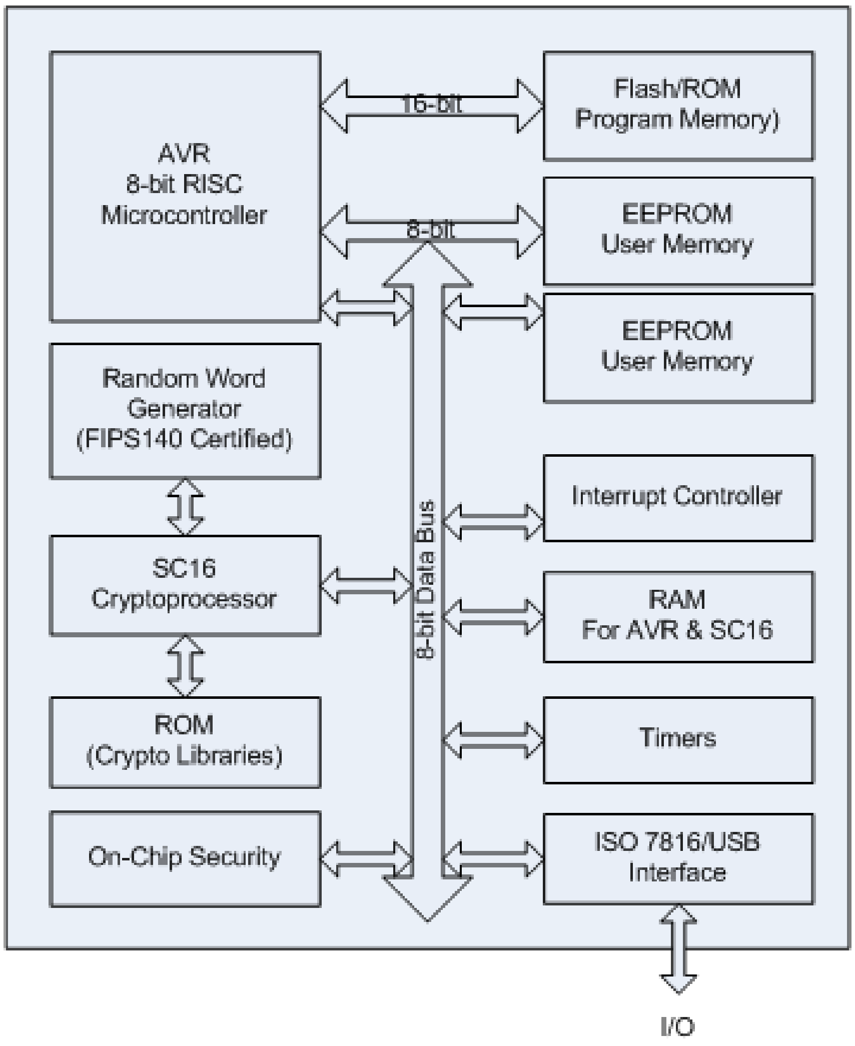


Figure 5.7: SecureAVR block diagram

Example: AT90SC288144RU

The AT90SC288144RU is a 8-/16-bit microcontroller with Read-Only Memory (ROM) program memory and EEPROM memory, based on the SecureAVR enhanced RISC architecture.

The AT90SC288144RU uses the SecureAVR architecture that allows the linear addressing of up to 8 MB of code and up to 16 MB of data, as well as a number of new functional and security features. It features 144 KB of EEPROM. The ability to map the EEPROM into code space allows parts of the program memory to be reprogrammed at runtime. Additional security features include logical scrambling on program data and addresses so that data and addresses are stored more randomly in the memory, in addition with power analysis countermeasures and a memory accesses controlled by a supervisor mode.

Key aspects of the 8/16-bit microcontroller AT90SC288144RU are 135 instructions (most executed in a single clock cycle), 288 KB ROM, 144 KB EEPROM, including 128 One-Time Programmable (OTP) bytes and 384-byte bit-addressable bytes, an RNG, hardware DES and 3DES, which are Differential Power Analysis (DPA) resistant and a checksum accelerator.

The device also offers dedicated hardware for protection against Simple Power Analysis (SPA), DPA, Simple Electromagnetic Analysis (SEMA), and Differential Electromagnetic Analysis (DEMA) attacks [KGO01, QS01], protection against physical attacks, a voltage monitor, and a secure memory management with access protection. More information on features and criteria of the

SecureAVR can be found in [Atmh] or [Atmg].

Power Management and Performance

By executing instructions in a single clock cycle, the AT90SC288144RU achieves a throughput close to 1 MIPS per MHz. Its architecture includes 32 general-purpose working registers directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle.

Currently, it seems there are no further documents on power consumption of the SecureAVR, however, it can be expected to be very low, since the SecureAVR is only used in very low power scenarios. In Smardcard scenarios, power-consumption is less important, since a Smartcard has a constant voltage/current supply.

Applicability in WSNs

Since the device is not designed as a sensor node, it is not used as microcontroller for sensor nodes. Thus, the SecureAVR seems not to be feasible in WSNs, despite the available security features protect some of the assets described in Section 3.1.1. One reason for this could be the absence of wireless or wired interfaces and the, compared to, e.g., the ARM Cortex-M0, lower system performance.

5.2.6 Trusted Sensor Node (TSN)

Due to the absence of devices equipped with cryptographic co-processors in combination with very low power consumption, IHP [Gmb] develops a device to be used as sensor node inside security critical WSNs. The following part gives an overview of this upcoming device.

Architecture Overview

Based on the Leon2-Architecture (see Section 5.2.2), the TSN is divided into a single-chip, which includes the processor, hardware co-processor, and additional memory. The node is build to be a bridge between IEEE 802.11 and IEEE 802.15.4 wireless technology. Therefore it is equipped with both types of radio modules.

To fulfil the security requirements, such as integrity and confidentiality (see Section 3.2), the TSN was upgraded, compared to the Leon2 reference design, with co-processors for AES, SHA1, Pseudo-Random Number Generator (PRNG), as well as ECC, which are used for encryption, decryption and signing of messages. Figure 5.8 illustrates the architecture of the TSN, core characteristics of the TSN are shown in Table 5.2.

Table 5.2: TSN: Core characteristics

Attribute	**Value**
Area	30 mm^2
Signal Ports	98
Power Ports	24
BIST/Scan Ports	5
Cache	2x4 KB
Maximum Frequency (MHz)	16
Core voltage (V)	2.5
Pad Voltage (V)	3.3

The main part of the sensor node is the TSN-processor. Additionally external memory, such as RAM or Flash complete the sensor node. The TSN supports two communication interfaces, namely 802.11 and 802.15.4.

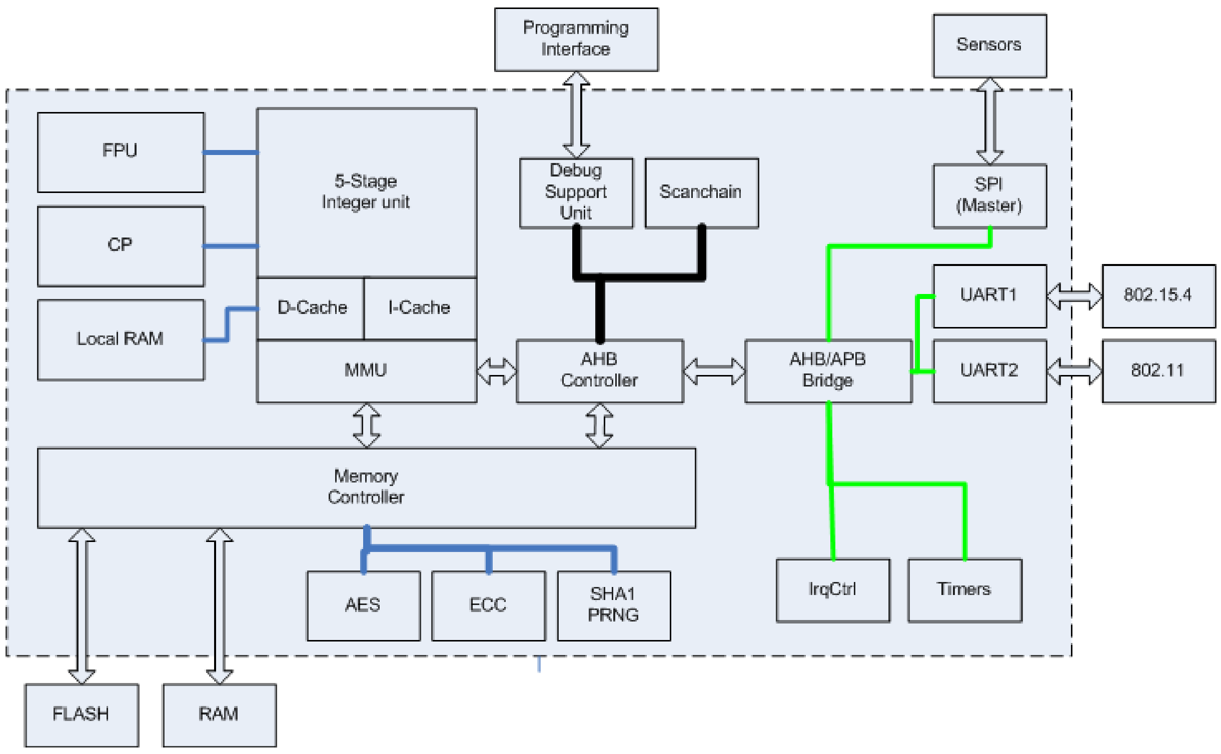


Figure 5.8: TSN block diagram

Memory The TSN supports 4 KB cache for commands and additional 4 KB cache for data. To minimize the chip size, the TSN does not support any on-chip flash memory but supports up to 16 MB external flash memory. Additionally, the TSN is equipped with 16 KB up to 16 MB RAM on the sensor board. Both RAM and flash are connected using the AMBA AHB-Bus and a memory controller using 32-bit words. Due to the low amount of I/O-pins, external memory can only be accessed using two 16-bit words, which means every load and store operation of a 32-bit value results in two accesses using 16-bit values.

Cryptographic co-processors The TSN supports an on-chip implementation of AES (AES 128 bit in Electronic Code Book (ECB)-Mode), ECC (ECC 233 bit), SHA1 and a PRNG. All three implementations are directly connected to the internal memory controller.

Power Management and Performance

The following Table 5.3 shows the energy consumption of the TSN, which is calculated in a way similar described like in Section 5.4.3.

Table 5.3: Power consumption and performance of the TSN

Component	Energy consumption	Runtime	Clock-cycles
SHA-1	8.12 mW	5125 ns	82
AES	3.94 mW	4875 ns	78
ECC	29 mW	822875 ns	13166
LEON	47.2 mW	about 6.25 ms	about 100000
whole TSN-system	95 mW	about 6.25 ms	about 100000

Applicability in WSNs

Compared with other architectures, e.g., the SecureAVR (see Section 5.2.5), the TSN is highly applicable in WSNs. It is equipped with cryptographic accelerators to increase the speed of the node and it consumes a low amount of energy.

5.2.7 Texas Instruments MSP430

- CC430 RF SoC Series

The MSP430 supports a wide range of hardware integrated peripherals like 10-/12 bit successive approximation ADC (SARADC), 16-bit Sigma Delta Analog-to-Digital Converter (ADC), UART, I^2C, or SPI. A complete list of available peripherals and interfaces can be found at [Texc].

Since there is a very huge spectrum of sub-architectures, we focus on the MSP430F5xx. It is the flash-based microcontroller family featuring low power consumption and performance up to 25 MIPS. It has an operation voltage range from 1.8V up to 3.6V, has an integrated power management module, including an internally controlled voltage regulator, and a wide range of memory options up to 256 kB.

Power Management and Performance

This next generation MSP430 microcontroller offers about 165 μA per MIPS, 2.5 μA consumption in standby and 0.1 μA in shut-down mode. The platform can wake up from standby in less the 5 μs. It supports up to 12 MHz at 1.8V, what can be increased up to 25 MHz. The MSP430 supports 1.8V flash erase and write, fail-safe and flexible clocking system, and up to 1 MB linear memory addressing [Texc].

Like the different types of MSP430 families, there are over 30 different types of sensor nodes in each family. Thus it is impossible to give a clear definition of performance and memory of this type of architecture. In this document we take two specific MSP430 sensors of the F5xx family, the MSP430F5529 and the MSP430F5438, compare both processors to each other and give a overview of their key features.

Example: MSP430F5529 and the MSP430F5438A

Table 5.4 lists the features of both nodes to introduce and give an example on what is possible in sensors using this type of architecture.

Table 5.4: Comparison of the MSP430F5529 and the MSP430F5438

	MSP430F5529	**MSP430F5438A**
Program (kB)	128	256
SRAM (kB)	8(+2 if USB is disabled)	16
I/O	63	83
16-bit Timers	4	3
Watchdog and RTC	yes	yes
Power Management	yes	yes
UART/LIN/IrDA/SPI	2	4
I2C/SPI	2	4
DMA	3ch	3ch
MPY	yes	yes
Comp_B	yes	yes
Temperature Sensor	yes	yes
ADC Ch/Res	16ch ADC 12A	16ch ADC 12A
Additional Features	USB	25 MIPS
Packages	80 PN	100 PZ, 113 ZQW
Price per 1kU	4.00	4.85

Applicability in WSNs

Together with the Atmel ATmega (MICAz [Cro]), which is not mentioned in this document, the Texas Instruments MSP430 is one of the current standards in the field of WSNs. It is equipped with a huge amount of peripheral interfaces and has a low amount of power consumption, thus it is a good solution as SNs.

5.3 Comparison Of Hardware Architectures Regarding Hardware Criteria

Regarding hardware criteria introduced in Section 5.1, the following tables illustrate how the considered architectures handle each of the considered criteria. Also a short conclusion of every comparison is given.

In most of the cases, the criteria strongly depend on the specific sensor node, using the mentioned hardware architecture, thus not all criteria could be analysed in detail.

5.3.1 Interfaces

Table 5.5 illustrates the supported interfaces of sensor nodes, based on each hardware architecture. Especially the peripheral interfaces strongly depend on the specific sensor and its architecture. As an example, the ARM Architecture introduced in Section 5.2.1 offers a huge amount of interfaces, and is the most flexible architecture mentioned in this study, compared to the SecureAVR introduced in Section 5.2.5, which offers almost no interface, due to the usage in only a specific microcontroller area.

Table 5.5: Interfaces of each hardware architecture

	ARM	**Leon2**	**Leon3**	**XMEGA**	**Sec.AVR**	**TSN**	**MSP430**
Ethernet	partial	yes	yes	no	no	yes	no
ZigBee	partial	partial	unknown	unknown	no	yes	partial
WLAN	partial	partial	unknown	unknown	no	yes	partial
UART	partial	yes	yes	yes	no	yes	yes
USB	partial	no	yes	no	yes	no	yes
SDIO	partial	no	no	no	no	no	no
GPIO	partial	no	no	no	no	no	no
SPI	partial	yes	no	yes	no	yes	yes
I2C	partial	no	no	yes	no	no	yes

5.3.2 Performance

Table 5.6 illustrates the performance criteria for each hardware architecture. In this analysis, it was not possible to compare the memory performance for each architecture, since it strongly depends on the implemented sensor node. Also, the processor speed depends on implementation-specific aspects such as clock speed, which depends on the power-management, and temperature requirements of the node. Regarding the performance of the cryptographic engine of the device, it can be said that if a device has a build-in hardware-accelerator, such as AES or DES, the overall speed of this architecture, using these specific operations, is strongly increased. In Section 5.4, a short comparison of common cryptographic operations implemented in each hardware architecture, is made, taking the TSN as an example (see Section 5.2.6), the clock-cycles of common cryptographic operations dramatically decreases with build-in accelerators (see Table 5.3). Since this analysis is based on the architectures, the speed of network-interfaces of each architecture also could not be estimated.

Table 5.6: Performance of each hardware architecture

	ARM	**Leon2**	**Leon3**	**XMEGA**	**Sec.AVR**	**TSN**	**MSP430**
Memory	1	1	1	1	1	1	1
Processor	high	high	high	medium	low	high	low
Crypto	medium	medium	medium	high	low	high	low
Network	1	1	1	1	1	1	1

1. unknown

5.3.3 Security Extensions

The majority of the analysed hardware architectures has no hardware security extension. Since the SecureAVR (see Section 5.2.5) is not used as sensor node in WSNs, the XMEGA architecture introduced in Section 5.2.4, together with some ARM architectures, are the only architectures with build-in security extensions to increase cryptographic speed or offer the possibility to separate untrusted code from trusted code (see Section 5.2.1). As a conclusion, it could be said that in the past security was a less important aspect in designing microcontroller architectures, however, more and more architectures are designed with a more clear focus to be used in security relevant areas. Currently, security of available sensors strongly depend on the additionally implemented security features of the sensor manufacturer. As an example, the TSN introduced in Section 5.2.6 implements security extensions on top of the Leon2 architecture, described in Section 5.2.2. Table 5.7 illustrates the implemented security extensions inside each hardware architecture.

Table 5.7: Security Extensions inside each hardware architecture

	ARM	**Leon2**	**Leon3**	**XMEGA**	**Sec.AVR**	**TSN**	**MSP430**
Isolation	partly	no	no	no	no	no	no
Booting	partly	no	no	no	unknown	no	no
Storage	partly	no	no	no	yes	no	no
RNG	no	no	no	no	partly	no	no
Crypto HW	partly	no	no	yes	partly	yes	no

5.3.4 Memory

Similar to performance and interfaces, the available amount and type of memory of the analysed hardware architectures strongly depends on the implemented sensor node and less depends on the architecture itself. Most of the architectures offer interfaces to access internal or external memory, but do not have a fixed amount of memory by design. Regarding the fault management, the Leon3 (introduced in Section 5.2.3) is the only architecture capable to provide this feature by design.

Table 5.8 illustrates all test criteria regarding memory of each hardware architecture.

Table 5.8: Memory aspects of each hardware architecture

	ARM	**Leon2**	**Leon3**	**XMEGA**	**Sec.AVR**	**TSN**	**MSP430**
Available Memory	dep.[1]	dep.[1]	dep.[1]	dep.[1]	dep.[1]	4KB [2], 0-16MB	dep.[1]
Memory Type	dep.[1]	PROM, SRAM, SDRAM	PROM, SRAM, SDRAM	Flash, EEPROM, SRAM	Flash, EEPROM, ROM,	Flash, RAM	unknown
Continuing next page							

Memory aspects of each hardware architecture - continued

	ARM	Leon2	Leon3	XMEGA	Sec.AVR RAM	TSN	MSP430
Fault Managem.	no	unknown	partly	no	unknown	no	no

1. depends on implementation
2. commands and data

5.3.5 Power Management

A sensor node with an efficient power management offers a better node lifetime, thus power management is one of the main aspects of a comparison of hardware architectures in WSNs. Section 5.4 analyses the energy efficiency of each mentioned architecture in detail, while running common cryptographic operations, such as AES and SHA1. Additionally, most of the hardware architectures offer their own power management capabilities that, if used intensively, also increase the node lifetime. However, the overall efficiency strongly depends on the sensor node itself, less on the architecture used inside the node. Due to the fact that this analysis is based on hardware architectures but not sensor nodes, there cannot be a clear conclusion about the power management of a specific sensor node used inside WSNs.

5.3.6 Assurance

The only hardware architecture that fulfils assurance requirements is the SecureAVR (introduced in Section 5.2.5). Accordingly, we can conclude that currently available architecture used in WSNs do not implement on-chip security features to fulfil common assurance requirements.

Table 5.9 illustrates all existing security standards of the considered hardware architectures.

Table 5.9: Assurance of each hardware architecture

	ARM	Leon2	Leon3	XMEGA	Sec.AVR	TSN	MSP430
Sec. standard	none	none	none	none	EAL5+	none	none

1. unknown

5.3.7 Additional Criteria

Depending on the performance and power management aspects of each architecture, a conclusion could be that sensor nodes with a focus on low power consumption use 16-bit architectures, while those with more features and more performance use 32-bit architectures. However, none of the analysed architectures was equipped with an DSP by design. Table 5.10 summarises all additional criteria of each considered hardware architecture.

Table 5.10: Additional criteria of each hardware architecture

	ARM	Leon2	Leon3	XMEGA	Sec.AVR	TSN	MSP430
DSP	no	no	no	no	no	no	no
Architecture	32-bit	32-bit	32-bit	8-/16-bit	16-bit	32-bit	16-bit
Manufacture	several	several	several	several	unkn.[1]	Germ.	several
WSN capable	partly	partly	partly	yes	no	yes	yes
Price	unkn.[1]	unkn.[1]	unkn.[1]	unkn.[1]	unkn.[1]	unkn.[1]	unkn.[1]

1. unknown

5.4 Energy efficiency

Energy efficiency is one of the most important aspects in WSNs or embedded devices in general, since an energy efficient sensor node consumes less energy, resulting in a longer node-life-time. Thus, this section describes the methodology used to estimate the energy consumption or energy efficiency of hardware architectures described in Section 5.2.

5.4.1 Methodology

In general, consumed energy is the product of time and power consumption. The power-consumption is based on 100% duty cycles and is sometimes provided by the manufacture of the specific hardware architecture. The time is the cycle-count of the operation to be measured. To give an easy example, we assume that a processor consumes 30 mW, running at 10 MHz. With these values, while operating 1 million cycles, the processor would consume:

$$1Million/10MHz * 30mW = 3mWs = 3mJ \tag{5.1}$$

In this analysis, we apply three steps to estimate the energy efficiency of each architecture as described in the following:

1. *Compile a piece of code:* We focus on the operations AES and SHA1. Publicly available example-programs using AES and SHA1 will be cross-compiled for the specific architecture. The detailed information how the source-code was compiled for each architecture, is described in the following hardware-specific sections. The source-code that was used for this comparison are the AES [Hooa, Hoob] and SHA1 [Hood] reference implementation of the Hoozi Resources team [Hooc], which is listed in the corresponding sections Section 5.4.4, Section 5.4.5, and Section 5.4.6. Each cross-compilation resulted in specific assembler code, generated to run on the targeted architecture.

2. *Sum-up each specific assembler command:* In this step, the cross-compiled source-code was summed-up by every single assembler operation. These assembler-operations are used in step three for the calculation of the clock-cycles.

3. *Calculation of the clock-cycles:* Based on the summed-up assembler-operation of the prior step, together with the list of necessary clock-cycles of each assembler-operation of each architecture, the total clock-cycles of the operations AES and SHA1 are calculated.

The application of this methodology represents the first creterion for the energy efficiency of each architecture. An additional, more complicated but also more detailed way to calculate the consumed energy is described in Section 5.4.3.

5.4.2 Energy Efficiency Analysis

In the following section, each architecture is considered regarding the described methodology in Section 5.4.1.

ARM-Architecture, especially Cortex-M0

The first step to estimate the energy efficiency of the Cortex-M0 processor is to compile the source-code listed in Section 5.4.4, Section 5.4.5 and Section 5.4.6. For this purpose, the ARM toolchain from CodeSourcery was used [Cod]. This toolchain provides a modified version of the gcc-4.4.1 compiler, and is capable to compile source-code for the ARM Cortex-M0 processor, introduced in Section 5.2.1. The source-code is compiled using the commands:

```
arm-none-eabi-gcc -S aes-dec.c -o aes-dec.cortex-m0
arm-none-eabi-gcc -S aes-enc.c -o aes-enc.cortex-m0
arm-none-eabi-gcc -S sha1.c -o sha1.cortex-m0
```

The second step includes a sum-up of every assembler command to count the quantity of each operation. According to the ARM Cortex-M0 technical reference manual [ARMd], the processor implements the ARMv6-M Thumb instruction set (Section 5.2.1), including a number of 32-bit instructions that use Thumb-2 technology (Section 5.2.1). Table 5.11 lists the ARM Cortex-M0 instructions and their cycle counts that are based on a system with zero wait-states. Taking these instructions and the summed-up assembler operations, Table 5.12 lists each type of operation with its quantity and the necessary clock-cycles. Moreover, the total amount of clock-cycles for AES-decryption, AES-encryption and SHA1 is listed.

Thus, as a first estimation of the energy efficiency, the ARM Cortex-M0 uses 718 clock-cycles for SHA1, 1484 clock-cycles for AES-encryption, and 3466 clock-cycles for AES-decryption, using assembler-code that was not specifically optimised for the ARM Cortex-M0 architecture.

Table 5.11 illustrates the instruction summary of the ARM Cortex-M0 architecture and gives an overview of the clock-cycles needed for each type of operation.

Table 5.11: ARM Cortex-M0 instruction summary

Operation	**Description**	**Assembler**	**Cycles**
Move	8-bit immediate	MOVS Rd, #<imm>	1
	Lo to Lo	MOVS Rd, Rm	1
	Any to Any	MOV Rd, Rm	1
	Any to PC	MOV PC, Rm	3
Add	3-bit immediate	ADDS Rd, Rn, #<imm>	1
	All registers Lo	ADDS Rd, Rn, Rm	1
	Any to Any	ADD Rd, Rd, Rm	1
	Any to PC	ADD PC, PC, Rm	3
	8-bit immediate	ADDS Rd, Rd, #<imm>	1
	With carry	ADCS Rd, Rd, Rm	1
	Immediate to SP	ADD SP, SP, #<imm>	1
	Form address from SP	ADD Rd, SP, #<imm>	1
	Form address from PC	ADR Rd, <label>	1
Subtract	Lo and Lo	SUBS Rd, Rn, Rm	1
	3-bit immediate	SUBS Rd, Rn, #<imm>	1
	8-bit immediate	SUBS Rd, Rd, #<imm>	1
	With carry	SBCS Rd, Rd, Rm	1
	Immediate from SP	SUB SP, SP, #<imm>	1
	Negate	RSBS Rd, Rn, #0	1
Multiply	Multiply	MULS Rd, Rm, Rd	1 or 32^a
Compare	Compare	CMP Rn, Rm	1
	Negative	CMN Rn, Rm	1
	Immediate	CMP Rn, #<imm>	1
Logical	AND	ANDS Rd, Rd, Rm	1
	Exclusive OR	EORS Rd, Rd, Rm	1
	OR	ORRS Rd, Rd, Rm	1
	Bit clear	BICS Rd, Rd, Rm	1
	Move NOT	MVNS Rd, Rm	1
	AND test	TST Rn, Rm	1

Continuing next page

ARM Cortex-M0 instruction summary - continued

Operation	**Description**	**Assembler**	**Cycles**
Shift	Logical shift left by immediate	LSLS Rd, Rm, #<shift>	1
	Logical shift left by register	LSLS Rd, Rd, Rs	1
	Logical shift right by immediate	LSRS Rd, Rm, #<shift>	1
	Logical shift right by register	LSRS Rd, Rd, Rs	1
	Arithmetic shift right	ASRS Rd, Rm, #<shift>	1
	Arithmetic shift right by register	ASRS Rd, Rd, Rs	1
Rotate	Rotate right by register	RORS Rd, Rd, Rs	1
Load	Word, immediate offset	LDR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	LDRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	LDRB Rd, [Rn, #<imm>]	2
	Word, register offset	LDR Rd, [Rn, Rm]	2
	Halfword, register offset	LDRH Rd, [Rn, Rm]	2
	Signed halfword, register offset	LDRSH Rd, [Rn, Rm]	2
	Byte, register offset	LDRB Rd, [Rn, Rm]	2
	Signed byte, register offset	LDRSB Rd, [Rn, Rm]	2
	PC-relative	LDR Rd, <label>	2
	SP-relative	LDR Rd, [SP, #<imm>]	2
	Multiple, excluding base	LDM Rn!, <loreglist>	$1+N^b$
	Multiple, including base	LDM Rn, <loreglist>	$1+N^b$
Store	Word, immediate offset	STR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	STRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	STRB Rd, [Rn, #<imm>]	2
	Word, register offset	STR Rd, [Rn, Rm]	2
	Halfword, register offset	STRH Rd, [Rn, Rm]	2
	Byte, register offset	STRB Rd, [Rn, Rm]	2
	SP-relative	STR Rd, [SP, #<imm>]	2
	Multiple	STM Rn!, <loreglist>	$1+N^b$
Push	Push	PUSH <loreglist>	$1+N^b$
	Push with link register	PUSH <loreglist>, LR	$1+N^b$
Pop	Pop	POP <loreglist>	$1+N^b$
	Pop and return	POP <loreglist>, PC	$4+N^c$
Branch	Conditional	B<cc> <label>	1 or 3^d
	Unconditional	B <label>	3
	With link	BL <label>	4
	With exchange	BX Rm	3
	With link and exchange	BLX Rm	3
Extend	Signed halfword to word	SXTH Rd, Rm	1
	Signed byte to word	SXTB Rd, Rm	1
	Unsigned halfword	UXTH Rd, Rm	1
	Unsigned byte	UXTB Rd, Rm	1
Reverse	Bytes in word	REV Rd, Rm	1
	Bytes in both halfwords	REV16 Rd, Rm	1
	Signed bottom half word	REVSH Rd, Rm	1
State change	Supervisor Call	SVC #<imm>	$-^e$
Continuing next page			

ARM Cortex-M0 instruction summary - continued

Operation	**Description**	**Assembler**	**Cycles**
	Disable interrupts	CPSID i	1
	Enable interrupts	CPSIE i	1
	Read special register	MRS Rd, <specreg>	4
	Write special register	MSR <specreg>, Rn	4
	Breakpoint	BKPT #<imm>	–[e]
Hint	Send event	SEV	1
	Wait for event	WFE	2[f]
	Wait for interrupt	WFI	2[f]
	Yield	*YIELD*[g]	1
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	4
	Data memory	DMB	4
	Data synchronization	DSB	4

a. Depends on multiplier implementation.
b. N is the number of elements.
c. N is the number of elements in the stack-pop list including PC and assumes load or store does not generate a hard-fault exception.
d. 3 if taken, 1 if not-taken.
e. Cycle count depends on core and debug configuration.
f. Excludes time spent waiting for an interrupt or event.
g. Executes as NOP.

Table 5.12: Quantity of assembler operations compiled for the ARM Cortex-M0

Operation	**Cycles**	**AES-dec**	**AES-enc**	**SHA1**
Move	1	1082	145	72
Add	1	375	104	67
Subtract	1	32	23	14
Subtract negate	1	4	4	4
Logical AND	1	325	43	8
Logical OR	1	0	0	3
Logical Exclusive OR	1	190	25	7
Load Multiple	1	35	34	6
Load Word, immediate offset	2	199	223	134
Load Byte, immediate offset	2	243	84	5
Store	2	113	96	70
Store multiple	1	31	28	6
Compare	1	22	22	12
Brunch conditional (taken)	3	21	21	10
Brunch unconditional	3	18	17	10
Brunch with exchange	3	9	8	3
Brunch with link	4	29	28	8
Sum of cycles		3466	1484	718

Leon2-Architecture

As for the ARM example, the first step included the compilation of the source code. For this purpose, the LEON Bare-C Cross Compilation System (BCC) from Gaisler Research [Gaib] has been used. It operates with a modified version of the gcc 3.4.4 and is capable to compile source-code for the Leon2-Architecture, introduced in Section 5.2.2, which is based on the SPARC V8 architecture. The source-code is compiled using the commands:

```
sparc-elf-gcc -S aes-dec.c -o aes-dec.SPARC
sparc-elf-gcc -S aes-enc.c -o aes-enc.SPARC
sparc-elf-gcc -S sha1.c -o sha1.SPARC
```

The second step includes a sum-up of every assembler operation to count their quantity. Compared to the energy-efficiency-analysis of the ARM Cortex-M0 (Section 5.4.2), the analysis of the Leon2-Architecture cannot be that detailed, since the timing in SPARC V8 architectures is strictly implementation-dependent [Inca]. Thus we use Table 5.13 as an approximation of cycle-counts of the Leon2-Architecture. It uses the cycles per instruction (assuming cache hit and no load interlock) and is based on the Leon2 Processor User's Manual [Res].

Table 5.13: Leon2 instruction summary and cycle counts

Instruction	**Cycles**	**AES-dec**	**AES-enc**	**SHA1**
JMPL (Jump and Link)	2	0	0	0
Double load	2	0	0	0
Single store	2	88	93	10
Double store	3	0	0	0
SMUL/UMUL (Sig./Uns. Int. Multiply)	(1/2/4/5/35)[1]	0	0	0
SDIV/UDIV (Sig./Uns. Int. Divide)	35	0	0	0
Taken Trap	4	87	84	24
Atomic load/store	2	0	0	0
All other instructions	1	2483	758	155
Sum of cycles		3007	1280	271

1. Depends on multiplier implementation.

Summarising this analysis, it can be said, that the Leon2 architecture uses 271 clock-cycles for SHA1, 1280 clock-cycles for AES-encryption, and 3007 clock-cycles for AES-decryption, using assembler-code that was not specifically optimised for the Leon2 architecture.

Leon3-Architecture

Since the Leon3-Architecture introduced in Section 5.2.3 is based on the Leon2-Architecture (Section 5.2.2), the analysis of clock-cycles is not necessary, since there is no difference on the instruction or architecture level that executes each operation. Both are based on a SPARC architecture and differ only by the fact that Leon3 supports a multi-core environment, whereas Leon2 is a single-core architecture.

Atmel AVR XMEGA

Considering to the three steps defined in Section 5.4.1, we compiled the source-code listed in Section 5.4.4, Section 5.4.5, and Section 5.4.6, using the cross-compiler avr-gcc, an open-source cross-compiler published by Atmel that includes a modified version of the avr-gcc 4.3.4 and is capable to compile code for the AVR XMEGA, described in Section 5.2.4. In this analysis, an

Ubuntu 9.10, x86-64 testing system is used, thus the pre-compiled package [AVRa, AVRb] had to be installed with the following command:

```
sudo dpkg --force-architecture -i avr-gcc-4.3.4-avrfreaks-09-mar-2010.deb
export PATH=/usr/local/avr/bin/:$PATH
```

After the successful installation, the source-code was compiled with the commands:

```
avr-gcc -mmcu=avrxmega3 -S aes-enc.c -o aes-enc.XMEGA
avr-gcc -mmcu=avrxmega3 -S aes-dec.c -o aes-dec.XMEGA
avr-gcc -mmcu=avrxmega3 -S sha1.c -o sha1.XMEGA
```

The parameter "-mmcu=avrxmega3" optimises the code for devices with an amount of flash memory between 8 KB and 64 KB, including more than 64 KB RAM. If necessary, the command "avr-gcc –target-help" gives an overview about the architectures, supported by the used compiler.

The resulting assembler-code is analysed according the AVR XMEGA clock-cycles of each assembler operation. Table 5.14 lists the CPU clock-cycles for all instructions of the AVR XMEGA. All information are based on the XMEGA Manual [Atmd].

Table 5.14: AVR XMEGA instruction summary

Instruction	**Description**	**Cycl.**	**AES-d**	**AES-e**	**SHA1**
ADD	Add without Carry	1	304	23	42
ADC	Add wit Carry	1	304	23	76
ADIW	Add Immediate to Word	2	49	48	18
SUB	Subtract without Carry	1	4	4	3
SUBI	Subtract Immediate	1	99	84	64
SBC	Subtract with Carry	1	60	4	4
SBCI	Subtract Immediate with Carry	1	98	82	64
SBIW	Subtract Immediate from Word	2	13	9	4
AND	Logical AND	1	0	0	16
ANDI	Logical AND with Immediate	1	112	0	8
OR	Logical OR	1	0	0	24
EOR	Exclusive OR	1	302	25	36
COM	One's Complement	1	0	0	5
INC	Increment	1	0	0	1
DEC	Decrement	1	0	0	3
TST	Test for Zero or Minus	1	2	3	2
CLR	Clear Register	1	88	4	10
RJMP	Relative Jump	2	20	22	14
RCALL	Relative Call Subroutine	2/3[1]	7	7	0
CALL	Call Subroutine	3/4[1]	27	27	9
RET	Subroutine Return	4/5[1]	9	8	2
CP	Compare	1	4	5	2
CPC	Compare with Carry	1	20	20	10
CPI	Compare with Immediate	1	16	15	8
SBRC	Skip if Bit in Register Cleared	1/2/3	0	0	3
BREQ	Branch if Equal	1/2	2	2	0
BRNE	Branch if Not Equal	1/2	7	6	2
BRGE	Branch if Greater or Equal, Sig.	1/2	6	6	10
BRLT	Branch if Less Than, Signed	1/2	11	11	2
Continuing next page					

AVR XMEGA instruction summary - continued

Instruction	**Description**	**Cycl.**	**AES-d**	**AES-e**	**SHA1**
MOV	Copy Register	1	307	25	16
MOVW	Copy Register Pair	1	534	96	81
LDI	Load Immediate	1	230	47	69
LDS	LoadDirectfrom data space	2^{12}	48	50	0
LD	Load Indirect	$1/2^{12}$	41	45	28
LDD	Load Indirect with Displacement	2^{12}	413	268	337
STS	Store Direct to Data Space	2^{1}	16	16	0
ST	Store Indirect	$1/2^{12}$	40	32	24
STD	Store Indirect with Displacem.	2^{1}	134	128	203
IN	In From I/O Location	1	24	22	16
OUT	Out To I/O Location	1	30	26	12
PUSH	Push Register on Stack	1^{1}	24	21	8
POP	Pop Register from Stack	2^{1}	34	31	8
LSL	Logical Shift Left	1	837	82	31
LSR	Logical Shift Right	1	0	0	2
ROL	Rotate Left Through Carry	1	1013	0	37
ROR	Rotate Right Through Carry	1	5	95	9
ASR	Arithmetic Shift Right	1	5	5	0
SWAP	Swap Nibbles	1	0	0	8
Sum of cycles			6114	2084	1940

1. Data memory accesses cycles assume internal memory accesses.
2. One extra cycle must be added when accessing internal SRAM.

Summarising this analysis, it can be said that the XMEGA architecture uses 1940 clock-cycles for SHA1, 2084 clock-cycles for AES-encryption, and 6114 clock-cycles for AES-decryption, using assembler-code that was not specifically optimised for the XMEGA architecture.

The XMEGA architecture is an excellent example that the performance and power-management extremely depends on the implementation of hardware and software. Referencing [Ott], it is possible to implement the AES algorithm highly optimised so that the AES-decryption only uses 4443 clock-cycles, build for an AVR microcontroller with a 8-bit architecture. Since the AVR XMEGA has a build-in AES accelerator, it is assured that the clock-cycles of the already optimised algorithm can be decreased in size and clock-cycles even more. Since this study did not use these highly-optimised assembler-codes, only the above mentioned estimated clock-cycles could be used.

Atmel SecureAVR

There is no public document on the clock-cycles of the Atemel SecureAVR (Section 5.2.5) available, thus an energy efficiency analysis could not be conducted in this study.

Texas Instruments MSP430

According to the three steps defined in Section 5.4.1, we compiled the source-code listed in Section 5.4.4, Section 5.4.5, and Section 5.4.6, using the cross-compiler mspgcc4, an open-source cross-compiler published on sourceforge [Und], which includes a modified version of the gcc 4.4.2 and using the following commands:

```
msp430-gcc -S sha1.c -o sha1.MSP430
msp430-gcc -S aes-enc.c -o aes-enc.MSP430
msp430-gcc -S aes-dec.c -o aes-dec.MSP430
```

The resulting assembler-code is analysed according the Texas Instruments MSP430 clock-cycles of each assembler operation. Table 5.15 lists the CPU clock-cycles for reset, interrupts, and subroutines, Table 5.16 lists the CPU cycles for all addressing modes of instructions with single-operand and Table 5.17 lists the CPU cycles for all addressing modes with multiple-operands. All these information are based on the MSP430x5xx Family User's Guide ([Texa]).

Table 5.15: Interrupt, return and reset cycles of the Texas Instruments MSP430

Operation	**Cycles**
Return from interrupt RETI	5
Return from subroutine RET	4
Interrupt request service (cycles needed before first instruction	6
WDT reset	4
Reset (RST/NMI)	4

Table 5.16: Instruction with single-operand cycles of the Texas Instruments MSP430

Adress Mode	**Cycl.(RRA,RRC, SWPB,SXT)**	**Cycles PUSH**	**Cycles CALL**	**Exam.**
Rn	1	3	4	SWPB R5
@Rn	3	3	4	RRC @R9
@Rn+	3	3	4	SWPB @R10+
#N	N/A	3	4	CALL #LABEL
X(Rn)	4	4	5	CALL 2(R7)
EDE	4	4	5	PUSH EDE
&EDE	4	4	6	SXT &EDE

Table 5.17: Instruction with single-operand cycles of the Texas Instruments MSP430

Source	**Destination**	**Cycles**	**Example**
Rn	Rm	1	MOV R5,R8
	PC	4	BR R9
	x(Rm)	4	ADD R5,4(R6)
	EDE	4	XOR R8,EDE
	&EDE	4	MOV R5,&EDE
@Rn	Rm	2	AND @R4,R5
	PC	4	BR @R8
	x(Rm)	5	XOR @R5,8(R6)
	EDE	5	MOV @R5,EDE
	&EDE	5	XOR @R5,&EDE
@Rn+	Rm	2	ADD @R5+,R6
	PC	4	BR @R9+
	x(Rm)	5	XOR @R5,8(R6)
	EDE	5	MOV @R9+,EDE
	&EDE	5	MOV @R9+,&EDE

Continuing next page

Instruction with single-operand cycles of the Texas Instruments MSP430 - continued

Source	Destination	Cycles	Example
#N	Rm	2	MOV #20,R9
	PC	3	BR #2AEh
	x(Rm)	5	MOV #0300h,0(SP)
	EDE	5	ADD #33,EDE
	&EDE	5	ADD #33,&EDE
x(Rn)	Rm	3	MOV 2(R5),R7
	PC	5	BR 2(R6)
	TONI	6	MOV 4(R7),TONI
	x(Rm)	6	ADD 4(R4),6(R9)
	&TONI	6	MOV 2(R4),&TONI
EDE	Rm	3	AND EDE,R6
	PC	5	BR EDE
	TONI	6	CMP EDE,TONI
	x(Rm)	6	MOV EDE,0(SP)
	&TONI	6	MOV EDE,&TONI
&EDE	Rm	3	MOV &EDE,R8
	PC	5	BR &EDE
	TONI	6	MOV &EDE,TONI
	x(Rm)	6	MOV &EDE,0(SP)
	&TONI	6	MOV &EDE,&TONI

Now the generated assembler-code was analysed and an assumption of used clock-cycles for AES and SHA1 was made. Table 5.18 lists the results of approximately used clock-cycles in the Texas Instruments MSP430 architecture.

Table 5.18: Quantity of assembler operations compiled for the Texas Instruments MSP430

AES-dec	AES-enc	SHA1
4284 cycles	1520 cycles	1307 cycles

Summarising this analysis, it can be said that the TI MSP430 architecture uses 1307 clock-cycles for SHA1, 1520 clock-cycles for AES-encryption, and 4284 clock-cycles for AES-decryption, using assembler-code that was not specifically optimised for the MSP430 architecture.

Trusted Sensor Node (TSN), based on Leon2

Since the TSN (introduced in Section 5.2.6) is based on the Leon2-Architecture (Section 5.2.2), a separated analysis of the energy efficiency based on the clock-cycles is not necessary. A precise list of used clock-cycles and consumed power is listed in Table 5.3. This list is made, using a more precise methodology introduced in Section 5.4.3.

Comparison of architectures

Figure 5.9, and Table 5.19 illustrate the used clock-cycles of each analysed architecture for the operations AES-decryption, AES-encryption and SHA1. As mentioned before, the Leon3 and the SecureAVR could not be analysed with regard to their used clock-cycles, since no further information was available.

Table 5.19: Comparison of clock-cycles of each hardware architecture

Architecture	AES-dec	AES-enc	SHA1
ARM Cortex-M0	3466 cycles	1484 cycles	718 cycles
Leon2	3007 cycles	1280 cycles	271 cycles
AVR XMEGA	6114 cycles	2084 cycles	1940 cycles
TSN	78 cycles	78 cycles	82 cycles
MSP430	4284 cycles	1520 cycles	1307 cycles

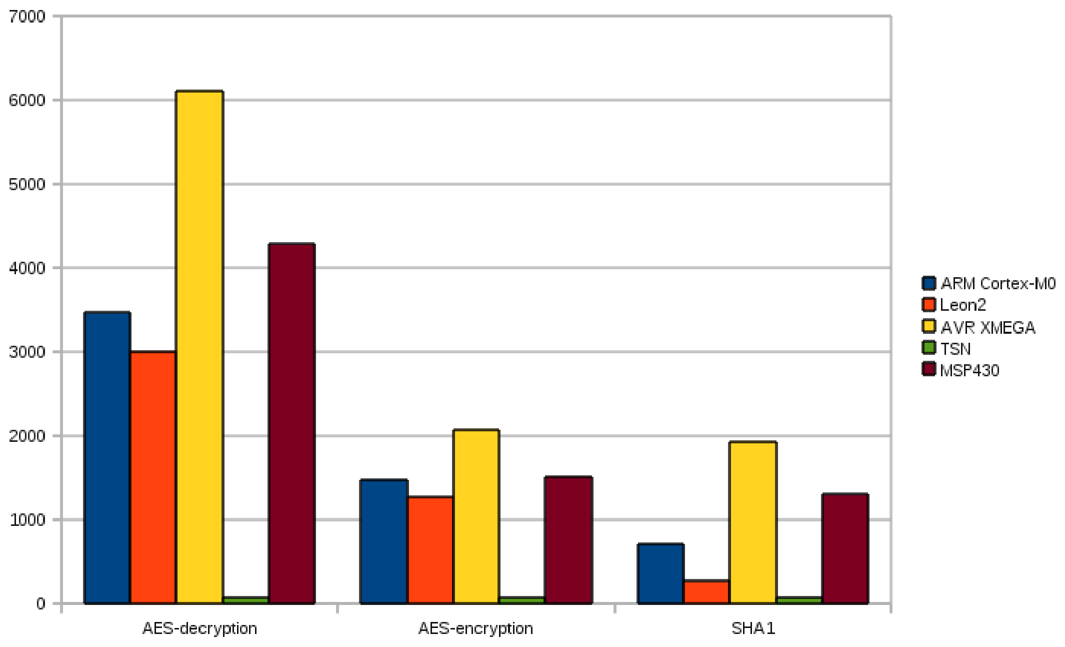


Figure 5.9: Comparison of clock-cycles of each hardware architecture

5.4.3 More precise methodology

The methodology described in Section 5.4.1 is a rough estimation of the real consumed energy of each architecture. A more precise methodology to calculate the energy of a specific code running on a hardware architecture is to simulate each architecture with all used gates and to execute the source-code inside these simulators. This leads to a very precise estimation of the energy-consumption of each architecture, since it is known, how much energy each gate consumes and, based on the VHDL-code of the used source-code, a precise calculation can be achieved.

Due to the fact that such an analysis needs very cost-intensive hardware- and software-simulators , such as Modul-SIM [Gra], power-analysis tools of Synopsys [Incb], or similar programs, this second methodology of energy consumption analysis could not be conducted in this study.

An example of this more precise methodology is the TSN (Section 5.2.6), where precise information about the energy consumption of AES, SHA1, and even ECC are available. The amount of used energy is listed in Table 5.3.

5.4.4 Used source-code of AES encryption

This section lists the source-code of AES encryption, used inside the energy efficiency analysis in Section 5.4.

```c
/*
******************************************************************
**       Advanced Encryption Standard implementation in C.      **
**       By Niyaz PK                                            **
**       E-mail: niyazpk@gmail.com                              **
**       Downloaded from Website: www.hoozi.com                 **
******************************************************************
This is the source code for encryption using the latest AES algorithm.
******************************************************************
*/

// Include stdio.h for standard input/output.
// Used for giving output to the screen.
#include<stdio.h>

// The number of columns comprising a state in AES. This is a constant
// in AES. Value=4
#define Nb 4

// The number of rounds in AES Cipher. It is simply initiated to zero.
// The actual value is recieved in the program.
int Nr=0;

// The number of 32 bit words in the key. It is simply initiated to zero.
// The actual value is recieved in the program.
int Nk=0;

// in - it is the array that holds the plain text to be encrypted.
// out - it is the array that holds the output CipherText after encryption.
// state - the array that holds the intermediate results during encryption.
unsigned char in[16], out[16], state[4][4];

// The array that stores the round keys.
unsigned char RoundKey[240];

// The Key input to the AES Program
unsigned char Key[32];

int getSBoxValue(int num)
{
        int sbox[256] =   {
        //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A
        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B
        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C
        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D
        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E
        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 }; //F
        return sbox[num];
}

// The round constant word array, Rcon[i], contains the values given by
// x to th e power (i-1) being powers of x (x is denoted as {02}) in the
// field GF(28)
// Note that i starts at 1, not 0).
int Rcon[255] = {
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
        0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
        0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
        0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
        0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
        0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
        0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
        0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
        0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
        0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
        0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
        0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
        0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
        0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
        0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb  };

// This function produces Nb(Nr+1) round keys. The round keys are used in
// each round to encrypt the states.
void KeyExpansion()
{
        int i,j;
        unsigned char temp[4],k;

        // The first round key is the key itself.
        for(i=0;i<Nk;i++)
        {
                RoundKey[i*4]=Key[i*4];
                RoundKey[i*4+1]=Key[i*4+1];
                RoundKey[i*4+2]=Key[i*4+2];
                RoundKey[i*4+3]=Key[i*4+3];
        }

        // All other round keys are found from the previous round keys.
        while (i < (Nb * (Nr+1)))
        {
                for(j=0;j<4;j++)
                {
                        temp[j]=RoundKey[(i-1) * 4 + j];
                }
                if (i % Nk == 0)
                {
                        // This function rotates the 4 bytes in a word to the left once.
                        // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

                        // Function RotWord()
                        {
                                k = temp[0];
                                temp[0] = temp[1];
                                temp[1] = temp[2];
                                temp[2] = temp[3];
                                temp[3] = k;
                        }

                        // SubWord() is a function that takes a four-byte input word and
                        // applies the S-box to each of the four bytes to produce an output word.

                        // Function Subword()
                        {
                                temp[0]=getSBoxValue(temp[0]);
                                temp[1]=getSBoxValue(temp[1]);
                                temp[2]=getSBoxValue(temp[2]);
                                temp[3]=getSBoxValue(temp[3]);
                        }

                        temp[0] =  temp[0] ^ Rcon[i/Nk];
                }
                else if (Nk > 6 && i % Nk == 4)
                {
                        // Function Subword()
                        {
                                temp[0]=getSBoxValue(temp[0]);
                                temp[1]=getSBoxValue(temp[1]);
                                temp[2]=getSBoxValue(temp[2]);
                                temp[3]=getSBoxValue(temp[3]);
                        }
                }
                RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
                RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
                RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
                RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
                i++;
        }
}

// This function adds the round key to state.
// The round key is added to the state by an XOR function.
void AddRoundKey(int round)
{
        int i,j;
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
}

// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
void SubBytes()
{
        int i,j;
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        state[i][j] = getSBoxValue(state[i][j]);
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
void ShiftRows()
{
        unsigned char temp;

        // Rotate first row 1 columns to left
        temp=state[1][0];
        state[1][0]=state[1][1];
        state[1][1]=state[1][2];
        state[1][2]=state[1][3];
        state[1][3]=temp;

        // Rotate second row 2 columns to left
        temp=state[2][0];
        state[2][0]=state[2][2];
        state[2][2]=temp;

        temp=state[2][1];
        state[2][1]=state[2][3];
        state[2][3]=temp;

        // Rotate third row 3 columns to left
        temp=state[3][0];
        state[3][0]=state[3][3];
        state[3][3]=state[3][2];
        state[3][2]=state[3][1];
        state[3][1]=temp;
}

// xtime is a macro that finds the product of {02} and the argument to
// xtime modulo {1b}
#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

// MixColumns function mixes the columns of the state matrix
// The method used may look complicated, but it is easy if you know
// the underlying theory.
// Refer the documents specified above.
void MixColumns()
{
        int i;
        unsigned char Tmp,Tm,t;
        for(i=0;i<4;i++)
        {
                t=state[0][i];
                Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i] ;
                Tm = state[0][i] ^ state[1][i] ; Tm = xtime(Tm); state[0][i] ^= Tm ^ Tmp ;
                Tm = state[1][i] ^ state[2][i] ; Tm = xtime(Tm); state[1][i] ^= Tm ^ Tmp ;
                Tm = state[2][i] ^ state[3][i] ; Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp ;
                Tm = state[3][i] ^ t ; Tm = xtime(Tm); state[3][i] ^= Tm ^ Tmp ;
        }
}

// Cipher is the main function that encrypts the PlainText.
void Cipher()
{
        int i,j,round=0;

        //Copy the input PlainText to state array.
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        state[j][i] = in[i*4 + j];

        // Add the First round key to the state before starting the rounds.
        AddRoundKey(0);

        // There will be Nr rounds.
        // The first Nr-1 rounds are identical.
        // These Nr-1 rounds are executed in the loop below.
        for(round=1;round<Nr;round++)
        {
                SubBytes();
                ShiftRows();
                MixColumns();
                AddRoundKey(round);
        }

        // The last round is given below.
        // The MixColumns function is not here in the last round.
        SubBytes();
        ShiftRows();
        AddRoundKey(Nr);

        // The encryption process is over.
        // Copy the state array to output array.
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        out[i*4+j]=state[j][i];
}
int main()
{
        int i;

        // Receive the length of key here.
        while(Nr!=128 && Nr!=192 && Nr!=256)
        {
                printf("Enter the length of Key(128, 192 or 256 only): ");
                scanf("%d",&Nr);
        }

        // Calculate Nk and Nr from the received value.
        Nk = Nr / 32;
        Nr = Nk + 6;



// Part 1 is for demonstrative purpose. The key and plaintext are given
// in the program itself.
//     Part 1: ********************************************************

        // The array temp stores the key.
        // The array temp2 stores the plaintext.
        unsigned char temp[16] = {0x00  ,0x01  ,0x02  ,0x03  ,0x04  ,0x05  ,
            0x06  ,0x07  ,0x08  ,0x09  ,0x0a  ,0x0b  ,0x0c  ,0x0d  ,0x0e  ,0x0f};
        unsigned char temp2[16]= {0x00  ,0x11  ,0x22  ,0x33  ,0x44  ,0x55  ,
            0x66  ,0x77  ,0x88  ,0x99  ,0xaa  ,0xbb  ,0xcc  ,0xdd  ,0xee  ,0xff};

        // Copy the Key and PlainText
        for(i=0;i<Nk*4;i++)
        {
                Key[i]=temp[i];
                in[i]=temp2[i];
        }

//           *********************************************************




// Uncomment Part 2 if you need to read Key and PlainText from the keyboard.
//     Part 2: ********************************************************
/*
        //Clear the input buffer
        flushall();

        //Recieve the Key from the user
        printf("Enter the Key in hexadecimal: ");
        for(i=0;i<Nk*4;i++)
                scanf("%x",&Key[i]);

        printf("Enter the PlainText in hexadecimal: ");
        for(i=0;i<Nb*4;i++)
                scanf("%x",&in[i]);
*/
//             ********************************************************


        // The KeyExpansion routine must be called before encryption.
        KeyExpansion();

        // The next function call encrypts the PlainText with the Key using AES algorithm.
        Cipher();

        // Output the encrypted text.
        printf("\nText after encryption:\n");
        for(i=0;i<Nk*4;i++)
                printf("%02x ",out[i]);
        printf("\n\n");
        return 0;
}
```

5.4.5 Used source-code of AES decryption

This section lists the source-code of AES decryption, used inside the energy efficiency analysis in Section 5.4.

```
/*
******************************************************************
**       Advanced Encryption Standard implementation in C.      **
**       By Niyaz PK                                            **
**       E-mail: niyazpk@gmail.com                              **
**       Downloaded from Website: www.hoozi.com                 **
******************************************************************
This is the source code for decryption using the latest AES algorithm.
******************************************************************
*/

// Include stdio.h for standard input/output.
// Used for giving output to the screen.
#include<stdio.h>

// The number of columns comprising a state in AES. This is a constant
// in AES. Value=4
#define Nb 4

// The number of rounds in AES Cipher. It is simply initiated to zero.
// The actual value is recieved in the program.
int Nr=0;

// The number of 32 bit words in the key. It is simply initiated to zero.
```

```c
// The actual value is recieved in the program.
int Nk=0;

// in - it is the array that holds the CipherText to be decrypted.
// out - it is the array that holds the output of the for decryption.
// state - the array that holds the intermediate results during decryption.
unsigned char in[16], out[16], state[4][4];

// The array that stores the round keys.
unsigned char RoundKey[240];

// The Key input to the AES Program
unsigned char Key[32];

int getSBoxInvert(int num)
{
        int rsbox[256] = {
        0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
        0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
        0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
        0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
        0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
        0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
        0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
        0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
        0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
        0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };
        return rsbox[num];
}

int getSBoxValue(int num)
{
        int sbox[256] = {
        //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };
        return sbox[num];
}

// The round constant word array, Rcon[i], contains the values given by
// x to th e power (i-1) being powers of x (x is denoted as {02}) in the
// field GF(2^8)
// Note that i starts at 1, not 0).
int Rcon[255] = {
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
        0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
        0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
        0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
        0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
        0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
        0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
        0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
        0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
        0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
        0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
        0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
        0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
        0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
        0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb  };

// This function produces Nb(Nr+1) round keys. The round keys are used in
// each round to decrypt the states.
void KeyExpansion()
{
        int i,j;
        unsigned char temp[4],k;

                // The first round key is the key itself.
        for(i=0;i<Nk;i++)
        {
                RoundKey[i*4]=Key[i*4];
                RoundKey[i*4+1]=Key[i*4+1];
                RoundKey[i*4+2]=Key[i*4+2];
                RoundKey[i*4+3]=Key[i*4+3];
        }

        // All other round keys are found from the previous round keys.
        while (i < (Nb * (Nr+1)))
        {
                for(j=0;j<4;j++)
                {
                        temp[j]=RoundKey[(i-1) * 4 + j];
                }
                if (i % Nk == 0)
                {
                        // This function rotates the 4 bytes in a word to the left once.
                        // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

                        // Function RotWord()
                        {
                                k = temp[0];
                                temp[0] = temp[1];
                                temp[1] = temp[2];
                                temp[2] = temp[3];
                                temp[3] = k;
                        }

                        // SubWord() is a function that takes a four-byte input word and
                        // applies the S-box to each of the four bytes to produce an
                        // output word.

                        // Function Subword()
                        {
                                temp[0]=getSBoxValue(temp[0]);
                                temp[1]=getSBoxValue(temp[1]);
                                temp[2]=getSBoxValue(temp[2]);
                                temp[3]=getSBoxValue(temp[3]);
                        }

                        temp[0] =  temp[0] ^ Rcon[i/Nk];
                }
                else if (Nk > 6 && i % Nk == 4)
                {
                        // Function Subword()
                        {
                                temp[0]=getSBoxValue(temp[0]);
                                temp[1]=getSBoxValue(temp[1]);
                                temp[2]=getSBoxValue(temp[2]);
                                temp[3]=getSBoxValue(temp[3]);
                        }
                }
                RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
                RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
                RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
                RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
                i++;
        }
```

```
}

// This function adds the round key to state.
// The round key is added to the state by an XOR function.
void AddRoundKey(int round)
{
        int i,j;
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
}

// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
void InvSubBytes()
{
        int i,j;
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        state[i][j] = getSBoxInvert(state[i][j]);
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
void InvShiftRows()
{
        unsigned char temp;

        // Rotate first row 1 columns to right
                temp=state[1][3];
        state[1][3]=state[1][2];
        state[1][2]=state[1][1];
        state[1][1]=state[1][0];
        state[1][0]=temp;

        // Rotate second row 2 columns to right
                temp=state[2][0];
        state[2][0]=state[2][2];
        state[2][2]=temp;

        temp=state[2][1];
        state[2][1]=state[2][3];
        state[2][3]=temp;

        // Rotate third row 3 columns to right
        temp=state[3][0];
        state[3][0]=state[3][1];
        state[3][1]=state[3][2];
        state[3][2]=state[3][3];
        state[3][3]=temp;
}

// xtime is a macro that finds the product of {02} and the argument to xtime
// modulo {1b}
#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

// Multiplty is a macro used to multiply numbers in the field GF(2^8)
#define Multiply(x,y) (((y & 1) * x) ^ ((y>>1 & 1) * xtime(x)) ^ ((y>>2 & 1)
    * xtime(xtime(x))) ^ ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^ ((y>>4 & 1)
    * xtime(xtime(xtime(xtime(x))))))

// MixColumns function mixes the columns of the state matrix.
// The method used to multiply may be difficult to understand for beginners.
// Please use the references to gain more information.
void InvMixColumns()
{
        int i;
        unsigned char a,b,c,d;
        for(i=0;i<4;i++)
        {
                a = state[0][i];
                b = state[1][i];
                c = state[2][i];
                d = state[3][i];

                state[0][i] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^ Multiply(d, 0x09);
                state[1][i] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^ Multiply(d, 0x0d);
                state[2][i] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^ Multiply(d, 0x0b);
                state[3][i] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^ Multiply(d, 0x0e);
        }
}

// InvCipher is the main function that decrypts the CipherText.
void InvCipher()
{
        int i,j,round=0;

        //Copy the input CipherText to state array.
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        state[j][i] = in[i*4 + j];

        // Add the First round key to the state before starting the rounds.
            AddRoundKey(Nr);

        // There will be Nr rounds.
        // The first Nr-1 rounds are identical.
        // These Nr-1 rounds are executed in the loop below.
        for(round=Nr-1;round>0;round--)
        {
                InvShiftRows();
                InvSubBytes();
                AddRoundKey(round);
                InvMixColumns();
        }

                // The last round is given below.
        // The MixColumns function is not here in the last round.
        InvShiftRows();
        InvSubBytes();
        AddRoundKey(0);

        // The decryption process is over.
        // Copy the state array to output array.
        for(i=0;i<4;i++)
                for(j=0;j<4;j++)
                        out[i*4+j]=state[j][i];
}
int main()
{
        int i;

        // Receive the length of key here.
        while(Nr!=128 && Nr!=192 && Nr!=256)
        {
                printf("Enter the length of Key(128, 192 or 256 only): ");
                scanf("%d",&Nr);
        }
        // Calculate Nk and Nr from the received value.
        Nk = Nr / 32;
        Nr = Nk + 6;


// Part 1 is for demonstrative purpose. The key and plaintext are given in
// the program itself.
//      Part 1: ********************************************************

        // The array temp stores the key.
        // The array temp2 stores the plaintext.
        unsigned char temp[32] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
                                  0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
        unsigned char temp2[16]= {0x69, 0xc4, 0xe0, 0xd8, 0x6a, 0x7b, 0x04, 0x30, 0xd8, 0xcd,
                                  0xb7, 0x80, 0x70, 0xb4, 0xc5, 0x5a};

        // Copy the Key and CipherText
        for(i=0;i<Nk*4;i++)
        {
                Key[i]=temp[i];
                in[i]=temp2[i];
        }

//           *********************************************************


// Uncomment Part 2 if you need to read Key and CipherText from the keyboard.
//      Part 2: ********************************************************
/*
        //Clear the input buffer
        flushall();

        //Recieve the Key from the user
        printf("Enter the Key in hexadecimal: ");
        for(i=0;i<Nk*4;i++)
                scanf("%x",&Key[i]);

        printf("Enter the CipherText in hexadecimal: ");
        for(i=0;i<Nb*4;i++)
                scanf("%x",&in[i]);
*/
//             ********************************************************


        //The Key-Expansion routine must be called before the decryption routine.
            KeyExpansion();

        // The next function call decrypts the CipherText with the Key using
        // AES algorithm.
            InvCipher();

        // Output the decrypted text.
        printf("\nText after decryption:\n");
        for(i=0;i<Nb*4;i++)
                printf("%02x ",out[i]);

        printf("\n\n");
        return 0;
}
```

5.4.6 Used source-code of SHA1

This section lists the source-code of SHA1, used inside the energy efficiency analysis in Section 5.4.

```
/*
******************************************************************
This source code is under development. Even though you can use it as
such, it is recommended you check back after a few days for an updated
version. The current version lacks descriptive comments also.
******************************************************************
*/


#include<stdio.h>
#include<string.h>
#include<malloc.h>
#include<math.h>
#include<stdlib.h>

#define rotateleft(x,n) ((x<<n) | (x>>(32-n)))
#define rotateright(x,n) ((x>>n) | (x<<(32-n)))

void SHA1(unsigned char * str1)
{
    unsigned long int h0,h1,h2,h3,h4,a,b,c,d,e,f,k,temp;

    h0 = 0x67452301;
    h1 = 0xEFCDAB89;
    h2 = 0x98BADCFE;
    h3 = 0x10325476;
    h4 = 0xC3D2E1F0;
```

```
unsigned char * str;
str = (unsigned char *)malloc(strlen((const char *)str1)+100);
strcpy((char *)str,(const char *)str1);

int current_length = strlen((const char *)str);
int original_length = current_length;
str[current_length] = 0x80;
str[current_length + 1] = '\0';

char ic = str[current_length];
current_length++;

int ib = current_length % 64;
if(ib<56)
    ib = 56-ib;
else
    ib = 120 - ib;

int i=0;
for(i=0;i < ib;i++)
{
    str[current_length]=0x00;
    current_length++;
}
str[current_length + 1]='\0';

for(i=0;i<6;i++)
{
    str[current_length]=0x0;
    current_length++;
}
str[current_length] = (original_length * 8) / 0x100 ;
current_length++;
str[current_length] = (original_length * 8) % 0x100;
current_length++;
str[current_length+i]='\0';

int number_of_chunks = current_length/64;
unsigned long int word[80];
int j=0;
int m=0;
for(i=0;i<number_of_chunks;i++)
{
    for(j=0;j<16;j++)
    {
        word[j] = str[i*64 + j*4 + 0] * 0x1000000 + str[i*64 + j*4 + 1]
            * 0x10000 + str[i*64 + j*4 + 2] * 0x100 + str[i*64 + j*4 + 3];
    }
    for(j=16;j<80;j++)
    {
        word[j] = rotateleft((word[j-3] ^ word[j-8] ^ word[j-14] ^ word[j-16]),1);
    }

    a = h0;
    b = h1;
    c = h2;
    d = h3;
    e = h4;

    for(m=0;m<80;m++)
    {
        if(m<=19)
        {
            f = (b & c) | ((~b) & d);
            k = 0x5A827999;
        }
        else if(m<=39)
        {
            f = b ^ c ^ d;
            k = 0x6ED9EBA1;
        }
        else if(m<=59)
        {
            f = (b & c) | (b & d) | (c & d);
```

```
                k = 0x8F1BBCDC;
            }
            else
            {
                f = b ^ c ^ d;
                k = 0xCA62C1D6;
            }

            temp = (rotateleft(a,5) + f + e + k + word[m]) & 0xFFFFFFFF;
            e = d;
            d = c;
            c = rotateleft(b,30);
            b = a;
            a = temp;
        }

        h0 = h0 + a;
        h1 = h1 + b;
        h2 = h2 + c;
        h3 = h3 + d;
        h4 = h4 + e;
    }

    printf("\n\n");
    printf("Hash: %x %x %x %x %x",h0, h1, h2, h3, h4);
    printf("\n\n");
}

int main()
{
    SHA1((unsigned char *)"The quick brown fox jumps over the lazy dog");
    return 0;
}
```

6 Security Mechanisms in Sensor Networks (M4)

In this chapter we combine and extend security mechanisms discussed in Chapter 4 to fulfill the security requirements of Section 3.2 based on the functional assumptions outlined in Section 3.1.2.

As a main contribution we propose a novel remote attestation scheme for sensor networks that combines software attestation with PUFs. We use this attestation scheme to enhance key management and allow collaborative attestation of nodes after software update, reboot or suspected compromise. For secure time synchronization and secure software update we can use existing techniques.

6.1 High-level Architecture

As depicted in in Figure 6.1, the TeSOS system can be represented as a layered set of modules. Modules in the Hardware and OS layers that are responsible for base functionality of the WSN and can be found in many standard WSN systems are shown in gray. In contrast, the security-relevant modules that are designed and integrated within the TeSOS project reside in the Security and Application layers, except for optional security hardware like PUF for cryptographic accelerators and the new "Attestation Mode" in the kernel layer. The attestation mode introduces a new mode in the operating system where most local processes and interaction with other systems is suspended to maximize local processing power and minimize unexpected interference.

In the following, we describe these additional security services and how they meet the expected security objectives.

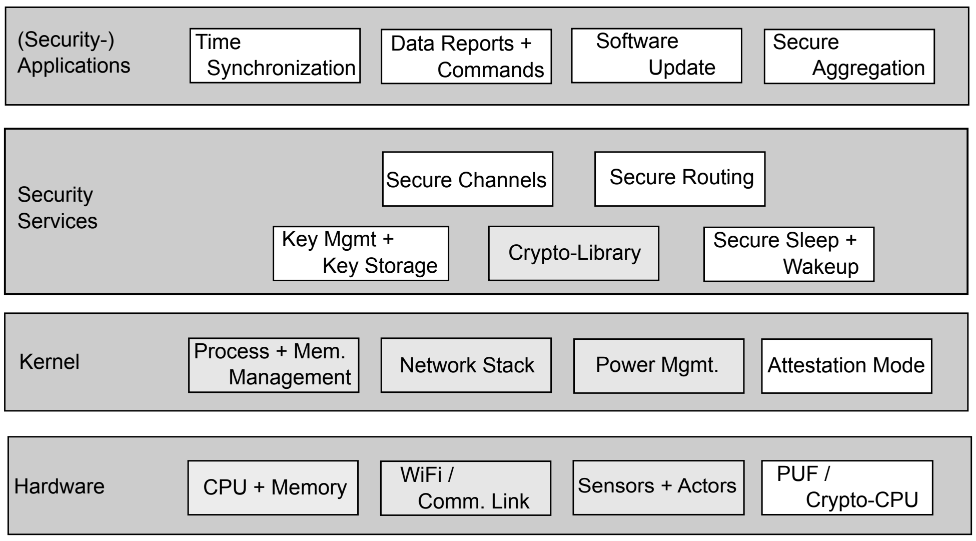


Figure 6.1: High-level TeSOS Architecture

6.2 Key Management

The functional requirements for key management in TeSOS are rather conservative in comparison to the assumptions made in the research literature. As detailed in Section 3.1.2, TeSOS Wireless Sensor Networks (WSNs) are comprised of at most $N = 1000$ nodes in a hierarchical topology where up to 50 Small Nodes (SNs) communicate mainly with their respective cluster head (Big Node, Big Node (BN)). The sensor nodes are installed statically and their location is known prior to deployment. However, TeSOS requires a limited resilience against failure of individual cluster heads (BNs), by supporting a mesh network of Small Nodes (SNs) to the next BN as a fallback solution.

Considering the above requirements, even a simple pair-wise key distribution that uses a priori information on node locations can be functionally sufficient: If we assume that the at most 50 SN that are reachable by each Cluster Head (CH) are the maximum amount of nodes that any node communicates with *directly*, each node requires only about $(50-1)\cdot 80/8/1024 \approx 0.5$ KBytes of key storage, plus node identifier information for each key. However, based on Section 3.2 one must also consider the following security requirements:

- **Secure Key Storage.** The sensor nodes require secure storage or a PUF to maintain confidentiality and integrity of encrypted data and messages despite hardware attacks.
- **Extendability.** The WSN must be extensible with additional nodes after deployment.
- **Revocation.** If a node is compromised or in unknown state, it must be possible to logically exclude it from the regular network to maintain integrity and confidentiality of the remaining network (revocation).
- **Forward and Backward Security.** The impact of compromised short- and long-term keys must be limited, i.e., it should not be possible to derive new session keys or long-term keys based on compromised older session keys, and if long-term keys are compromised it should not be possible to compromise session keys.

6.2.1 Symmetric vs. Asymmetric Key Management

A major question is whether the key management should rely on symmetric or asymmetric long-term (authentication) keys and/or asymmetric key exchange methods such as Ephemeral Diffie-Hellman (EDH). The following major differences between symmetric and asymmetric key management can be identified:

- **Scalability.** In symmetric systems, two parties can only communicate if they share a common secret key. However, for resilience against key compromise and unambiguous authentication of peers, every pair of nodes should use an independent common shared secret. As a result, the required key storage per node rises linearly in the number of peers. By contrast, key exchange based on asymmetric long-term authentication keys only requires each node to store its own key pair and the public key of a common certification (or key distribution) authority. A peer's public key is signed by the certification authority and the signature is verified during key exchange to authenticate the peer. Another advantage over symmetric key systems is that networks can be easily extended by having the certification authority sign the key pairs of new participants. In symmetric schemes, network extension is only possible if the existing network has a priori knowledge on the new participant's secret key. This is typically implemented at the cost of resilience, by introducing a global master secret. Alternatively, it would be possible to maintain a (limited) reserve of spare pairwise secrets on all participants.

Hence, symmetric key management is usually only practical in "closed" systems, where the size of the overall system can be estimated in advance and non-repudiation, i.e., unambiguous attribution of data to its sender, is not required. As the overall amount of pair-wise keys rises quadratically in the number of participants, symmetric key management requires some form of master key or other assumptions to reduce the required key storage in scenarios with many participants.

- **Forward and Backward Security.** Forward security considers the security of old session keys after the long-term keys that were involved in negotiation of the session keys have been compromised. To prevent an adversary from deriving all previous session keys based on compromised long-term keys, an additional secret must be involved in the key negotiation that is not available to the adversary even after key compromise. The property is typically achieved with an ephemeral Diffie-Hellman exchange as part of the session key negotiation, where the private ephemeral Diffie-Hellman keys are deleted after the negotiation is finished. Forward security is also possible with a purely symmetric key exchange, by replacing the long-term shared secret with a hash chain and periodically replacing the current secret key such that the previous secret cannot be reconstructed [BY03]. However, this method requires all involved peers to be synchronized with respect to their current position in the hash chain, and it must be hard for an adversary to desynchronize any two legitimate communication partners.

  Backward security considers the security of future session and long-term keys in face of compromised older session keys. Backward security is easily achieved by applying a cryptographic one-way functions (hash functions) on the session key before using it to protect the communication.

- **Resilience.** Key distribution in symmetric key management systems implies that all keys are also known to the Key Distribution Center (KDC). However, many ID-based encryption schemes also suffer from this so-called *key escrow* and even in non-ID-based asymmetric key schemes, the certification authority can forge certificates and launch man-in-the-middle-attacks. The problem can be mitigated by procedural measures, for example, by introducing multiple certification authorities and requiring a valid peer key to be signed by all of them independently. Similarly, the KDC in a symmetric or identity-based scheme can and should delete any knowledge on third party's pair-wise keys.

- **Performance.**

  Symmetric authentication and encryption algorithms typically require significantly less computational resources than the respective asymmetric alternatives. For the SN, asymmetric cryptography is therefore generally not practical due to message size and computational constraints. Even for BN asymmetric cryptography is only practical if hardware acceleration can be provided to reduce the computational cost.

Due to the closed nature of the considered sensor network and its rather low requirements on scalability, a symmetric key distribution scheme is functionally sufficient. Due to the assumption that nodes are not compromised in the deployment and upgrade phases, a protocol such as Localized Encryption and Authentication Protocol (LEAP) can provide the required scalability and extendibility of the network while maintaining resilience against partial network compromise. Simple counters can be used for forward security since nodes typically do not change their communication partners, mitigating the problem of desynchronization. However, the assumption of secure deployment and upgrade is rather uncommon and may not be practical in some situations. In the following we thus present two alternative solutions, one solution based on LEAP that is purely symmetric and one that combines LEAP with identity-based Elliptic Curve Cryptography (ECC) for additional robustness against key compromise.

6.2.2 A Forward-Secure Symmetric Key Management with Key Refresh

To meet the functional requirements and security goals of TeSOS, we suggest to combine the LEAP key management scheme [ZSJ03] with forward secure encryption based on one-way functions [BY03]. Optionally, remote attestation can be used for secure on-demand key refresh and re-synchronization of nodes. PUF-based remote attestation is introduced in later Section 6.5 and is modeled here simply as an oracle O_{AT} that yields a fresh symmetric secret shared between the sensor node and the base station.

As explained in Section 4.2.1, LEAP exploits a limited time frame t after deployment in which the adversary has not yet succeeded to compromise any node. Within this time frame, all nodes can thus be equipped with a master key K_m and negotiate link keys with node discovered neighbors as $K_B = h(K_m, ID_B)$ and $K_{AB} = f_{K_B}(ID_A, ID_B, N_1, N_2)$, where $h(\cdot), f(\cdot)$ are cryptographic one-way functions, A, B are the public identities of the involved sensor nodes broadcasted during network setup phase and N_1, N_2 are fresh random nonces. The resulting key K_A is the device key of A and K_{AB} the pairwise key shared between A and B. The master key K_m is purged from memory after time t, which is by design larger than the duration of the network setup phase, ensuring maximum resilience to node compromise once t is elapsed. As described in [ZSJ03], the scheme also supports group-keys and extensions of the network with new nodes. Note that in general, $K_{AB} \neq K_{BA}$. So if both, A and B, are in possession of K_m (e.g., immediately after deployment), then concurrent key negotiations must detected and the two parties must agree on which of the negotiated keys to use (e.g., by interpreting the node IDs as integers and canceling the key negotiation that was initiated by the node with the lower ID).

We introduce forward and backward security into this scheme by using the pairwise shared key K_{AB} as the root of a one-way key chain $(K_1, K_2, ..., K_N)$, where $K_i = f(K_i - 1)$ and $K_1 = K_{AB}$. To assure synchronization of the intervals i in a loosely coupled WSN with sleepy participants, we let the interval changes be triggered actively, by sending an authenticated command to the respective peer after another successful communication event occurred, to indicate the interval transition. An adversary may attempt to drop such indications or their acknowledgment, resulting in permanent desynchronization and potential DoS. Hence, peers must maintain a cache for the previously active keys until a successful authenticated interaction based on the new session key occurred. Keys from old intervals should be securely overwritten to guarantee forward security. In the event of desynchronization attacks, e.g., a man-in-the-middle repeatedly discarding certain notification messages, the nodes should report the problem to the Base Station (BS). In any case, a given symmetric key should never be used once ciphertext collisions become possible.

The protocol for key setup is depicted in Figure 6.2 and is equivalent with the standard LEAP protocol from [ZSJ03]: A and B generate pairwise shared keys K_{AB} and K_{BA} based on K_m and chose one of them as pairwise master secret. Note that, in later a WSN Upgrade phase, the older node (B, in Figure 6.2) has deleted K_m and only knows K_B. Still, both nodes can generate a common key K_{AB} for as long as A has not deleted its K_m.

The authors of LEAP also propose an extension to their scheme to increase the robustness against key compromise. Specifically, multiple instances of different keys K_m can be used over distinct phases t_i in the life time of the WSN. A node A added in phase t_i is equipped with the corresponding K_{m,t_1} to work in the scheme as described above, and additionally with device keys K_{A,t_j}, for all other life phases $t_j \neq t_i$. We can generalize this scheme to let the BS disclose additional master keys K_{m,t_i} to specific nodes A (e.g., a cluster head) after deployment, using the oracle O_{AT}. The mechanism can also be used to recover from compromised master and node keys as well as whenever key chains become desynchronized. The refresh protocol using on the remote attestation oracle O_{AT} is illustrated in Figure 6.3.

For sensor node revocation, we follow the approach of LEAP [ZSJ03] to let the BS announce revoked nodes by broadcasting their public identities (ID_A, ID_B in the above example) to all nodes in the network, which then delete all link and group keys established with that node. Since master keys K_{m,t_i} are deleted within a time frame where no compromise (and thus, no revocation)

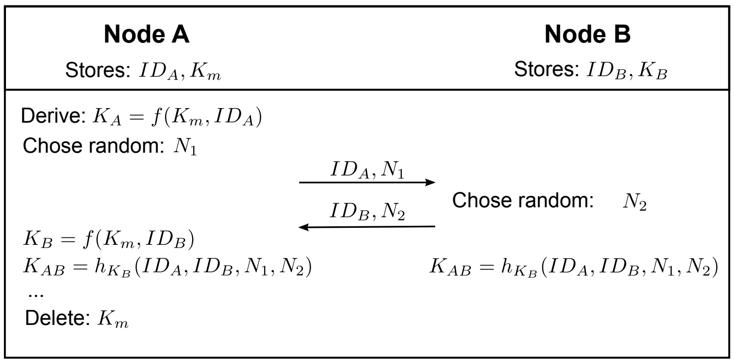


Figure 6.2: LEAP Key Setup Phase: Pairwise keys can be generated as long as either party has knowledge of K_m. After time t, deletion of K_m results in full resilience against node compromise

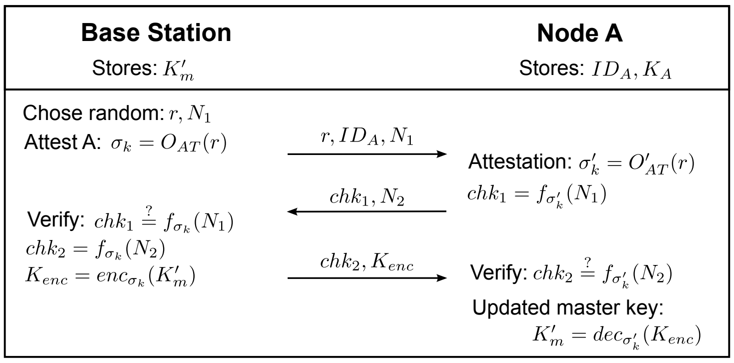

Base Station
Stores: K'_m
Node A
Stores: ID_A, K_A
Chose random: r, N_1
Attest A: $\sigma_k = O_{AT}(r)$
r, ID_A, N_1
Attestation: $\sigma'_k = O'_{AT}(r)$
$chk_1 = f_{\sigma'_k}(N_1)$
chk_1, N_2
Verify: $chk_1 \overset{?}{=} f_{\sigma_k}(N_1)$
$chk_2 = f_{\sigma_k}(N_2)$
$K_{enc} = enc_{\sigma_k}(K'_m)$
chk_2, K_{enc}
Verify: $chk_2 \overset{?}{=} f_{\sigma'_k}(N_2)$
Updated master key:
$K'_m = dec_{\sigma'_k}(K_{enc})$

Figure 6.3: LEAP Key Refresh Phase: After recovery from compromise, desynchronization of nodes or for re-initialization of nodes after undeployment, the remote attestation protocol (O_{AT}) can be used to securely distribute a new master key K'_m

occurs, the deletion of all link keys is sufficient to prevent revoked nodes from participating in the secure WSN. Additionally, since new keys K_{m,t_i} may be selectively disclosed to specific nodes or are temporarily available for newly deployed nodes when extending the WSN, these systems must also be informed of the current set of revoked sensor nodes.

6.2.3 Hybrid Key Management for Heterogeneous Sensor Networks

As we also note in Section 4.2.1 ID-based cryptography is well-suited for WSNs [ODL+07]. In the following we describe a combination of asymmetric and symmetric key exchange algorithms such that cluster heads communicate securely without requiring globally secure deployment and upgrade phases, while the key generation between cluster head (BN) and cluster members (SN) is done with the more efficient LEAP protocol.

Specifically, we choose a modification of the Arazi-Qi identity-based Diffie-Hellman exchange as proposed in [HUW11] to generate fresh keys between the BN. Communication between BN and SN is can then be protected by LEAP schemes deployed local to the respective sensor node cluster. As shown in Figure 6.4, each BN (including the Base Station (BS)) is equipped with the elliptic curve parameters $E(q, G)$, where q is the group modulus and G is the generator of E mode q, a global public key ("root certificate") R and the node identity ID. The KDC then generates public and private keys U, x for each sensor node, where they are verified and stored. Note that in contrast to LEAP, the KDC is not identical with the base station. Instead, the base station is treated as one of the BNs. After deployment, the cluster heads can use the key pairs to establish pair-wise keys securely, as illustrated in Figure 6.5.

The integration with LEAP, to allow secure communication between BN and SN, is straight-

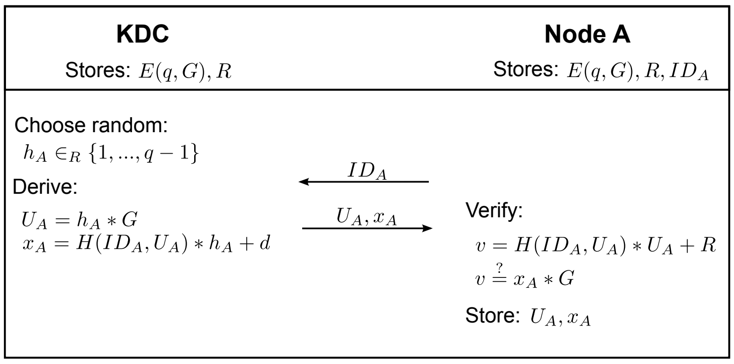

KDC
Stores: $E(q,G), R$
Node A
Stores: $E(q,G), R, ID_A$
Choose random:
$h_A \in_R \{1, ..., q-1\}$
Derive:
$U_A = h_A * G$
$x_A = H(ID_A, U_A) * h_A + d$
ID_A
U_A, x_A
Verify:
$v = H(ID_A, U_A) * U_A + R$
$v \overset{?}{=} x_A * G$
Store: U_A, x_A

Figure 6.4: Key generation for the modified Arazi-Qi scheme.

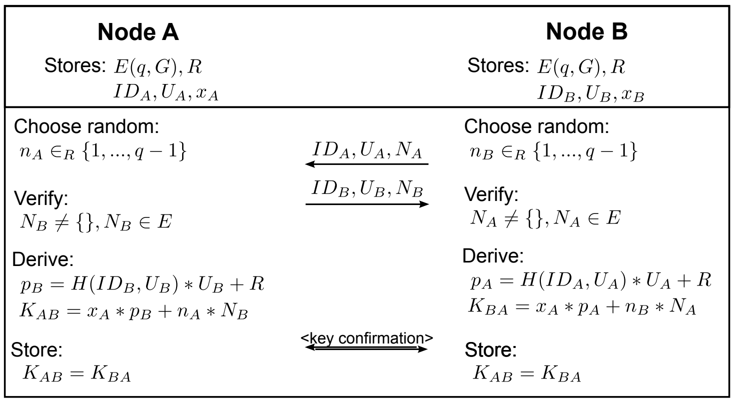


Figure 6.5: Ephemeral key agreement with the modified Arazi-Qi scheme.

forward: Since TeSOS assumes that the location of nodes is known in advance (see *WSN.Deployment* in Section 3.1.2), we can simply apply LEAP to the individual clusters by providing the cluster head (BN) and all cluster nodes (SN) of the same cluster c with the same uniformly random chosen LEAP master key K_m^c. To allow SN to fall back on alternative cluster heads (see Sections 3.1.2, 6.6), SN of different clusters must also be able to communicate securely. For this purpose, we propose to use the respective cluster heads of the two SN to distribute a new shared symmetric master key on demand. The LEAP protocols outlined in Section 6.2.2 can then be applied to the individual clusters, when one of the communication partners is an SN while the Arazi-Qi scheme (cf. Figure 6.5) is used if both communication participants are BNs.

6.2.4 Secure Key Storage

TeSOS nodes must provide secure key storage to meet the confidentiality and authenticity objectives for transferred messages and local data in face of hardware attacks. This particularly interesting for the LEAP protocol, where it is critical that the long-term keys cannot be extracted within time frame t.

Unfortunately, cryptographic modules that provide secure storage are not commonly available and likely too expensive for SNs and possibly also BNs. By contrast, Physically Unclonable Functions (PUFs) are already integrated in some embedded systems, although not commonly available yet. Physically Unclonable Functions (PUFs) are hardware functions that exploit intrinsic inaccuracies in chip production, and thus more cost-efficient by design.

Hence, we propose to adopt PUF-based key storage [TB05]: Instead of directly storing long-term keys such as K_m or K_{AB} in memory, only a PUF challenge (p_{ch}) and helper data (p_h) are stored instead of the actual key material. To access long-term keys, the PUF is then queried with p_{ch}. The PUF response is fed into a fuzzy extractor together with helper data p_h to derive the original secret key. During manufacturing, the memory that holds the PUF challenge is physically surrounded by the PUF. As a result, the PUF challenge cannot be extracted from the hardware without destroying the PUF, which in turn renders the extracted data useless.

To also mitigate the problem of software attacks, long-term keys should only be kept in memory while they are needed and securely overwritten afterwards [Gut01].

We emphasize that K_m must be protected in both, the purely LEAP-based scheme as well as the hybrid ECC/LEAP scheme. However, the hybrid keying scheme of Section 6.2.3 is less vulnerable to compromise since K_m is split in multiple localized cluster keys K_m^c. On the other hand, if the location and neighborhood of sensor nodes is known before deployment, this can also be done for the simple LEAP-based scheme.

6.2.5 Broadcast Authentication

Secure broadcast authentication must not only assure the authenticity of messages but also prevent replay and impersonation attacks. We identify two major approaches for secure broadcast authentication, μ-TESLA and asymmetric cryptographic signatures:

μ-TESLA. As described in Section 4.2.1, μ-TESLA uses delayed disclosure of symmetric authentication keys and loose time synchronization between all participants [PSW+01, LN03a, LN04]. The delayed disclosure prevents which is based only on symmetric receivers from impersonating the sender while still allowing a (delayed) authentication of genuine broadcast messages. Unfortunately, the scheme is very vulnerable to denial of service attacks, since receivers must buffer all received messages until the next respective authentication key is disclosed. Time intervals can be shortened, but as a result the time synchronization becomes tighter and it becomes more expensive for sender and receiver to maintain and validate the authentication keys.

Asymmetric Signatures. Signatures allow the verification of messages without possession of the secret key, and are thus by design more suitable for (broadcast) authentication of messages. However, asymmetric cryptography is generally much more expensive with regards to the required computation and communication efforts. We identify three options that can make broadcast authentication based on asymmetric cryptography more suitable for TeSOS:

- *Rabin-Williams Signatures.* As an optimization of the standard RSA signatures, Rabin-Williams signatures allow highly efficient signature verification that amounts to a simple modular squaring of the message. Other optimizations allow the message size, which is typically at least as large as the modulus, to be reduced by half [Ber08]. As a result, it becomes feasible to verify signatures in software. Signature creation at the BS is only slightly more expensive, however, the BS is assumed to be a rather powerful, laptop-class device. The main disadvantage of this approach is that it is bound to RSA, which is generally more resource intensive and produces longer message sizes than Elliptic Curve Cryptography (ECC). If an asymmetric pair-wise key establishment system is deployed with hardware support or other modifications, broadcast authentication with Rabin-Williams signatures cannot benefit from them.

- *Hardware Acceleration.* By requiring sensor nodes to support certain base operations in hardware, their energy and time consumption can be reduced drastically. In fact, optimal hardware acceleration for specific cryptographic operations can make the cost for individual operations almost negligible. However, such hardware extensions increase production cost and are often not available for low-cost devices (SNs).

- *Delegated Authentication.* To reduce the cost of signature verification, it may be viable to delegate authentication from the less resourceful leaves of the WSN (SNs) to the more powerful Cluster Heads (CHs). In this case, SNs must trust their respective CH to authenticate messages correctly and to not become compromised. In turn, the SN are not required to authenticate asymmetric signatures but can use pair-wise shared keys for much more efficient Message Authentication Codes (MACs). Since the pair-wise shared key is only known to the individual SN and their respective CH, the SN has assurance that messages are not spoofed as long as the CH is secure. The disadvantages of this approach are slightly higher communication and computation costs for pairwise authentication of messages, and the higher risk of compromised CHs.

To provide basic replay protection, a counter can be included in each message such that the receiver can compare the counter value and drop older messages. If the WSN supports basic time synchronization, a time stamp can be used instead of the counter to provide freshness of the message and support the time synchronization.

6.3 Data Channel Security

The TeSOS WSN features data flows typical for monitoring applications, where all data is collected by a base station, or distributed from a base station to all nodes in the network. In other words, the TeSOS network may have node-to-BS and BS-to-node data links, but never node-to-node communication patterns.

Providing end-to-end data security in networks with node-to-node communication is challenging in large WSNs, as this requires storing a large number of pair-wise keys [CP03]. However, it is rather straightforward for networks with data flows starting or ending in BS, as it requires each node to store only keys shared with the BS.

The general requirements for end-to-end data security are confidentiality, integrity, authenticity, freshness and availability (as defined in Table 3.1, page 17). In the following, we discuss end-to-end data security in the following application scenarios: (i) Collection of data reports; (ii) distribution of base station commands; and (iii) super distribution of code images.

6.3.1 Collecting Data Reports

Data reports include measurements and event reports collected by sensor nodes. Data reports are sent from individual sensor nodes to a base station, typically by means of unicast transmissions. The standard mechanism to ensure integrity and authenticity of unicast messages are MACs calculated over the message using a pair-wise key shared between the reporting sensor and the BS. Protection against replay attacks can be achieved using time stamps (see Section 6.7) or simple counters. When confidentiality of data reports is desired, messages must also be encrypted.

Availability of data reports is the most challenging requirement since we must assume that compromised nodes exist in the network. Compromised nodes can report non-existing events or refuse to forward messages. The state-of-the-art approach to tolerate such interference is to add redundancy. For instance, the underlying routing protocol can provide multiple routes to the destination (e.g., INSENS [DHM06a]). To mitigate reporting of false events, multiple nodes sensing the same event can also generate joint reports (e.g., [YYY$^+$05, RLZ08]). Both techniques increase intrusion tolerance of the network, however, at the cost of increased resource consumption (time and energy). Depending on requirements of a particular application on availability, none, both of these techniques of either of them can be integrated into the final system design.

Aggregated data reports. Data collected from sensor nodes can be merged by aggregation nodes to produce compact small output data that is forwarded to the destination, rather than sending individual data items. Aggregation nodes process collected data, e.g., apply actual aggregation functions to data, e.g., average, minimum or maximum (“lossy aggregation”), or compress several small reports into a single message in order to reduce protocol overhead ("lossless aggregation").

In the TeSOS WSN, BNs can adopt the role of data aggregators as they are the CHs and data reports of cluster nodes are always routed through the respective CH. Also, BNs possess more processing power and energy reserves and can thus afford the additional computing and communication loads.

Aggregating data reports reduces communication overhead, but also enables new attacks. An adversary may try to falsify the result of the aggregation output generated by each cluster, and to make the BS accepting false aggregation results. The easiest way to achieve such an attack would to be compromise the aggregating node and then generate arbitrary data reports. This is equivalent to compromising a significant portion of sensors of a cluster and supplying a big amount of bogus readings but requires much less efforts.

A number of algorithms has been proposed to secure data aggregation process [CPS06, YWZC08, HPP+07, TG08, BLM07, BLMB07]. Early protocols [CPS06, YWZC08] are not robust against network faults, as a single node failure results in rejection of a whole aggregation result. Protocols [HPP+07, TG08] are able to detect and localize fault or malicious nodes, however, at the penalty of high communication costs. The most suitable aggregation protocol for the TeSOS WSN is Secure Aggregation Protocol for Cluster-Based (SAPC) proposed by Bekara et al. [BLM07, BLMB07]. SAPC is intended for clustered sensor networks and is resilient to compromised aggregation nodes. In SAPC, the aggregation is computed and authenticated by the aggregating node and approved by all members of the cluster. Each cluster member observes data reported by other cluster members, calculates the aggregated value and computes MAC (with a key shared with the BS) over it. MACs are sent to a cluster head and included into a final report produced by a cluster head and transferred to a base station. The BS verifies the authenticity of the aggregation results calculated by each cluster member, and accepts the result if it is confirmed by the majority of cluster members.

Analysis. SAPC cannot be directly applied to the TeSOS network due to differences in assumptions on a network layout. Firstly, the SAPC network model assumes that all sensors within the cluster are placed within one-hop range and thus are able to hear broadcast messages of each other. In contrast, sensor nodes in a TeSOS cluster are within the radio range of CH but may not be within communication range of all other members of the cluster (see assumption WSN.Topology, Section 3.1.2). Secondly, SAPC does not consider the fall-back mechanism required in TeSOS where data can be delivered through alternative CHs. In both cases, cluster members may not be able to hear messages of all the nodes that contributed into the aggregation value. These differences can be addressed by (i) establishing direct radio range among cluster members where possible and (ii) not aggregating reports from SNs located beyond radio range, but delivering them individually, especially in case of SNs reporting via fall-back CHs. Note that SAPC requires broadcast authentication so that sensor nodes can receive and validate their peer's data reports broadcast messages and act as witness. SAPC proposes to use a MAC, calculated with a key from on one-way key chain. Specifically, the sender generates a sequence of one-hash chain keys $\{K_0, K_1, ..., K_n\}$, such that $K_{i-1} = H(K_i)$, where $i = 1...n$. A first commitment K_0 is distributed among all broadcast receivers, e.g., sent via authenticated unicast messages. The message is authenticated with MAC calculated with K_i, which is also included into a broadcast message. The broadcast recipient can authenticate the key by verifying $K_{i-1} = H(K_i)$. Each key is used only once for authentication, and each broadcast receiver only accepts the first message authenticated with K_i. A major drawback of this approach is that the broadcast can be intercepted, such that arbitrary messages can be sent based on the disclosed current key K_i. On the other hand, digital signatures are likely too expensive and μ-TESLA incurs rather large management overhead.

6.3.2 Distribution of Base Station Commands

In TeSOS, a base station can send commands to regulate and control network operation. For instance, it can request a particular data report, or change time interval for automated data reporting. The commands can be addressed to individual nodes, groups of nodes (e.g., a cluster), or to the whole network. Typically, multicast transmissions in WSNs are implemented as a filtered broadcast. To prevent replay of BS commands, a time stamp or counter value can be included and checked by the receiver counters synchronized among a sender and a receiver. Integrity and authenticity can be ensured by MAC calculated over the message with a pair-wise key shared by a base station and by a node the message is addressed to, or using one of the broadcast authentication schemes outlined in Section 6.2.5.

6.3.3 Secure Code Super Distribution

A typical application scenario that requires transmission of large amounts of data is the update sensor node program code with a new so-called code image. This process is usually carried out in a super-distribution fashion, i.e., the new code images are recursively distributed to all nodes in the network. The mechanism requires security measures to ensure the integrity, freshness and authenticity of a new code image. However, due to the size of the transferred data it is also highly susceptible to Denial of Service (DoS) attacks because the corruption of a single piece of the code image results in rejection of whole image due to authentication failure.

Hence, a stateful-verifier τ-time signature scheme was proposed which is based on purely cheap cryptographic primitives (i.e., hash function) and does not require time synchronization [UWB09]. When the entire code image is authenticated with only a single signature, this solution is susceptible to DoS attacks and transmission errors, since it is not possible to localize the corrupted part of the image. Thus, retransmission of entire code image would be required even if small block of the code image is lost or fault.

To address this problem, modern designs apply two major techniques: The first technique aims to divide the entire code image into pages and authenticates individual pages rather than the entire image [LGN06]. The hash of each page is included in the previous page, while the hash image of the first page is signed and included in a signature packet. However, their page size is larger than the size of the wireless package, thus the package cannot be authenticated on the fly, but only when the entire code page is received. Hence the approach is still vulnerable to DoS attacks. A second technique to improve DoS-resilience of secure code update was proposed based on Merkle hash trees [DHM06b]. The Merkle hash tree is used to allow each packet to be immediately authenticated upon receipt. However, this approach requires transmission of a Merkle hash tree for every page, thus increases communication costs and slows down code propagation.

The Seluge code super-distribution protocol builds up on earlier works but has an optimization to allow immediate authentication of each packet upon receipt without disrupting the efficient code image propagation [HNLD08]. They include hash image of each packet in the corresponding packet on the previous page, while Merkle hash tree is used to facilitate the authentication of the hash images of the packets in the first page.

Another line of research in the area of secure code distribution builds security mechanisms on Fountain Codes. Fountain Codes [Lub02] are designed to maintain high efficiency, in terms of protocol overhead, when transmitting small packets over lossy channels. Fountain Codes were adopted for super distribution in WSN and extended with resource-aware security extensions that allow to authenticate the source packets efficiently and almost on the fly[RZS+08, BHUW09]. Another extension to reduce protocol overhead integrates fuzzy control theory [MHU+09].

Analysis. Seluge [HNLD08] is a de-facto standard for a secure code super distribution in WSNs. However, it is designed for non-hierarchical homogeneous networks, where sensor nodes feature similar hardware and software. To be used in TeSOS network, it has to be adopted to a heterogeneous structure and a hierarchical network architecture.

For an update of a code image of BNs, unmodified Seluge can be used, because (i) BS and BNs can be seen as a non-hierarchical homogeneous network, and (ii) BNs are able to verify signatures made by a base station over the new image. However, the code update by SNs is more challenging. First of all, SNs reside on the second layer of network hierarchy, thus Seluge should be modified to be applicable for hierarchical networks. Second, SNs are not able to verify signatures made by BS, as they do not support asymmetric cryptographic operations.

The first issue can be addressed as following: a new code image for SNs can be first distributed among BNs, where stored in the external memory. As a next step, it can be distributed within the cluster. When all the nodes within the cluster feature the same functionality, the code update within the cluster can be performed by means of broadcast transmissions.

Although BNs can verify signature of BS over a new code image of SNs, it is not desirable to delegate signature verification to untrusted cluster heads. When compromised, a BN can compromise the rest of the cluster by replacing a code image with a malicious one. To enable a code image authentication by SNs, we propose usage of authentication tokens generated by a base station. This works like following: A base station generates authentication tokens for each SN in the network. The token is a MAC over a new code image (or, more precisely, over a first page of a code image) calculated with a pair-wise key shared by BS and a particular SN. The tokens are distributed among SNs such that they are able to authenticate the new code image. Without the valid token, the image update procedure is denied.

An alternative way to authenticate a first page of a new code image would be to use μ-TESLA. This would reduce network traffic (a single broadcast instead of multiple unicasts addressed to each SN in the network). However, this would enable an adversary to launch DoS attacks, as data authenticated with μ-TESLA scheme can be validated only after a certain delay.

6.4 Secure Boot and Wakeup

While Secure Boot is a well-established concept in mobile and entertainment systems, its application to sensor networks must additionally consider the strong hardware constrains on sensor network hardware: Continuous reboot or wakeup events may be used by an adversary to mount denial of sleep attacks (see Section 4.1.2). Furthermore, sensor nodes do not typically feature the required secure storage and wireless communication with trusted parties is expensive.

6.4.1 When to Boot: Secure Sleep Cycles

There are two major approaches to address the problem of sensor node wake-ups and reboots. One is to use authenticated commands from the base station, cluster head or other authorized parties, while the other is to schedule wake-up or reboot events in advance or based on sensory input (environmental events).

Authenticated Commands. Authenticated commands require not only a key infrastructure but also that the receiver can receive and validate the message before reacting to it. In case of wake-up commands, this only makes sense if the command can be validated in an operation mode that requires significantly less power. Since radio and CPU must be active, this approach only seems suitable for BN or nodes with additional peripherals that require large amounts of energy. If the WSN nodes are custom built, the approach of [SBS02, FH09] may be viable where a secondary low-power radio with integrated message authentication is used to boot or wake up the main hardware.

Scheduled Wake-Up. Periodic or event-based wake-up of devices is highly cost efficient, as it works with the standard timer hardware on existing sensor nodes. However, it also complicates the overall operation of the WSN as nodes are not reachable on demand, requiring caching and time synchronization. Furthermore, the scheduling of wake-up events is complicated if measurements should be reported to the base station quickly: While the sensor nodes can often be build such that the actual sensor can trigger a wake-up event for the local device, the delivery of the sensor measurement to other nodes is limited by the availability of intermediate nodes.

Analysis. The possibilities and limits of sleep cycles are highly dependent on the application of the sensor network. A mixture of scheduled wake-ups and authenticated commands for fine-tuning of the current sleep schedule appears promising to maintain flexibility for commodity hardware. If fast reporting of events is required, the wakeup of SN should be triggered by the sensor. For quick propagation of messages to the base station, the BN should then support a secondary low-power radio or wake up frequently, possibly listening for additional authenticated wake-up commands from peers.

6.4.2 What to Boot: Secure and Authenticated Boot

Secure Boot solutions typically require a secure hardware model that can be used to iteratively validate code that is loaded during bootstrapping. A typical implementation would authenticate all code with a digital signature before the code itself is executed. However, although first sensor hardware with hardware-accelerated cryptographic algorithms are available, secure and cost-efficient key storage is currently only (assumed to be) possible with PUFs.

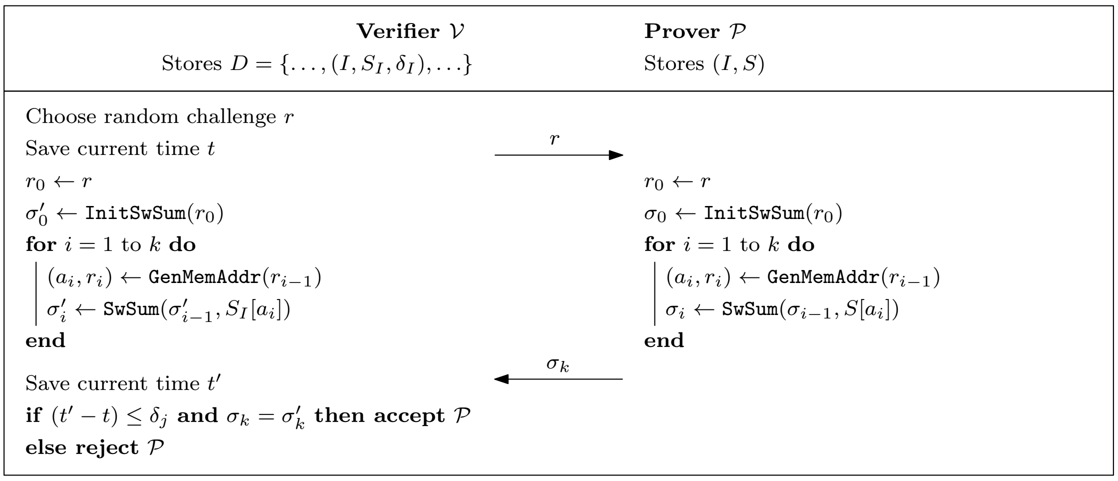
A Secure Boot solution for sensor networks requires a bootloader in ROM that can verify a digital signature of the currently stored OS image. The public key used for verification must be stored in ROM or linked to the PUF and its corresponding challenge and helper data stored in ROM, such that it cannot be exchanged by the adversary. Furthermore, ROM and signature verification as well as the memory interfaces must be protected against hardware attacks, to prevent the adversary from exchanging the OS image on the fly, after signature verification, or manipulating the ROM.

The alternate Authenticated Boot approach can be realized with additional hardware support, such as a Trusted Platform Module (TPM), or based on the more complex and computationally expensive software attestation scheme we describe in later Section 6.5.

Analysis Secure boot in face of strong hardware attacks appears impractical and costly, as the adversary has many attack vectors to manipulate program code. The merit of secure boot is also mitigated by the fact that the adversary can simply add additional nodes to the WSN or replace existing ones. In contrast, PUF-based key storage and remote attestation mitigate such attacks and also detect strong attacks involving hardware manipulation of benign nodes. We thus recommend to apply only rudimentary secure boot techniques, such as locally verifying and decrypting the software image before boot-up. On SNs, the required symmetric verification keys should be bound to the PUF for security against certain hardware attacks and on BN, an asymmetric verification key (i.e., public key of the BS) should be stored in ROM, if available.

6.5 Remote Attestation

Remote attestation allows to (re-)establish trust in remote systems, i.e., it provides assurance to a verifier $\mathcal{V}$ (e.g., the WSN base station) on the current (software) state of a remote device (prover $\mathcal{P}$). This allows to validate correct operation of WSN nodes, i.e., if it is executing the expected code or if undesired code was introduced by an adversary. It is useful in cases where unclear loss in service quality (correctness of measurements, bad network connection in specific areas of the network) is experienced or after recovery from possible software compromise (e.g., remote software update and reboot).



```
Verifier $\mathcal{V}$                                   Prover $\mathcal{P}$
Stores $D = \{\ldots, (I, S_I, \delta_I), \ldots\}$      Stores $(I, S)$

Choose random challenge $r$
Save current time $t$                 --- $r$ --->
$r_0 \leftarrow r$                                       $r_0 \leftarrow r$
$\sigma'_0 \leftarrow$ InitSwSum$(r_0)$                  $\sigma_0 \leftarrow$ InitSwSum$(r_0)$
for $i = 1$ to $k$ do                                    for $i = 1$ to $k$ do
    $(a_i, r_i) \leftarrow$ GenMemAddr$(r_{i-1})$            $(a_i, r_i) \leftarrow$ GenMemAddr$(r_{i-1})$
    $\sigma'_i \leftarrow$ SwSum$(\sigma'_{i-1}, S_I[a_i])$  $\sigma_i \leftarrow$ SwSum$(\sigma_{i-1}, S[a_i])$
end                                                      end
                                      <--- $\sigma_k$ ---
Save current time $t'$
if $(t' - t) \leq \delta_j$ and $\sigma_k = \sigma'_k$ then accept $\mathcal{P}$
else reject $\mathcal{P}$
```

Figure 6.6: Concept of software attestation

Unfortunately, existing remote attestation schemes require a trusted (hardware) component on every deployed sensor, which is expensive to protect against hardware attacks. The alternative software attestation schemes [SLS+05, SLP+06] on the other hand require that the verifier has a "direct", i.e., implicitly authenticated, connection to the prover, which is impractical for many WSN deployments. In the following, we describe an attestation scheme that provides true remote attestation by combining existing software attestation with PUFs [SSW11].

6.5.1 Combined Hardware/Software Attestation

A *software attestation* scheme is a two-party protocol between an *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$, where $\mathcal{V}$ requires the assurance that $\mathcal{P}$ is in a trusted (software) *state* S, i.e., that $\mathcal{P}$ is not running any malicious software. Typically, $\mathcal{P}$ is an embedded device with constrained computing capabilities (e.g., a wireless sensor node), whereas $\mathcal{V}$ is a more powerful computing device (e.g., a laptop). $\mathcal{V}$ can simulate any algorithm that can be executed by $\mathcal{P}$ and maintains a database D containing the identity I and the exact hard- and software configuration of each prover $\mathcal{P}$.

All known software attestation schemes follow the general concept illustrated in Figure 6.6. Abstractly, software attestation exploits the computational limits of $\mathcal{P}$ to assure that nothing else than a specific algorithm can be executed within a specific time frame δ. This algorithm is designed as a self-checksumming algorithm that also measures some additional parts of the software state of the system. Its execution time is optimized to the respective platform such that forging of the computed checksum, i.e., manipulating the checksum algorithm such that modifications to the measured system state are not reflected in the checksum, is not possible within the expected running time of the unmanipulated checksum algorithm. By measuring the delay between sending the attestation challenge and receiving the corresponding response from $\mathcal{P}$, $\mathcal{V}$ thus gains assurance on the software state of $\mathcal{P}$.

Existing software attestation schemes are implemented by iteratively applying a simple checksum function to specific memory blocks, merging their content into the current checksum state. To prevent efficient memory-remapping attacks, the order of memory blocks is typically chosen pseudo-randomly by a software-PRNG that is part of the overall software attestation. The PRNG is seeded based on the attestation challenge, which makes the overall software attestation procedure unpredictable to the adversary, preventing pre-computation attacks and also guaranteeing freshness of the attestation response.

In contrast to existing purely software-based attestation schemes, our solution provides assurance to the verifier that the attestation response was actually computed by the original hardware of the prover. This prevents the adversary from outsourcing the checksum computation to another, possibly more powerful device, or spoofing a different hardware configuration to forge the checksum in time.

Specifically, our scheme incorporates a characteristic *hardware checksum* in each iteration of the software attestation routine to generate a huge amount of additional input into the attestation routine that (1) cannot be generated by any other device except the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ and (2) cannot be transferred to an external device without significantly increasing the overall time required for the attestation procedure, such that the response of $\mathcal{P}$ to $\mathcal{V}$ is rejected at $\mathcal{V}$.

We achieve (1) by instantiating the hardware checksum procedure with a PUFs. In particular, in use cases like wireless sensor networks, where provers are deployed in hostile environments and are subject to hardware attacks, the tamper-evidence property of PUFs can ensure that modifications to the hardware of an prover can be detected. Property (2) is achieved by creating a strong interdependency between hardware and software checksum so that they cannot be parallelized efficiently and by ensuring that the overall amount of data exchanged between the algorithms is sufficiently high to prevent guessing or successive execution within the time frame allowed by $\mathcal{V}$.

We exploit that an adversary $\mathcal{A}$ is forced to use the external (wired or wireless) interfaces of $\mathcal{P}$ to transmit the hardware checksum results to a colluding device that is not bound by the computational performance of $\mathcal{P}$ and may thus compute a forged software attestation checksum in time. However, due to the limited transfer speed of these interfaces, we can ensure that $\mathcal{A}$ cannot obtain all hardware checksum results in time. On the other hand, an uncompromised node can execute the software checksum in time and use the typically much faster internal interfaces of $\mathcal{P}$ to exchange data between hardware and software checksums during attestation. Although an adversary may attempt to also access such faster internal interfaces of $\mathcal{P}$ at runtime, such an attack is significantly more complex, in particular when they are protected by tamper-evident hardware (e.g., a coating PUF [TSŠ+06, vMKT06] that is distributed over the surface of the integrated circuits of $\mathcal{P}$).

Unfortunately, the verification of the hardware checksum by the verifier $\mathcal{V}$ is not straightforward. To be compliant with existing software attestation schemes, $\mathcal{V}$ must be able to predict the outputs of the hardware checksum while this must be infeasible for the adversary $\mathcal{A}$. To achieve this, we discuss different approaches and instantiations of our protocol and evaluate their communication complexity and security. Hereby, we focus on solutions based on PUFs. However, predicting or verifying the output of a PUF to an unknown input is generally infeasible. Hence, the integration of PUFs into software attestation protocols is challenging and requires careful consideration.

Algorithm 1 illustrates the proposed overall attestation scheme. We abstract the pseudo-random address generator of the software attestation scheme illustrated in Figure 6.6 as a PRNG and merge additional random data from the `HwSum` into its internal state at runtime, e.g., using bitwise XOR. This way, each inclusion of output from `HwSum` also influences all future iterations of the `SwSum`, increasing the entanglement between the two functions.

6.5.2 Practical Considerations

In the following, we discuss some practical issues and options when deploying a PUF-based remote attestation scheme. For security analysis and additional details we refer to [SSW11].

Key Establishment

An important practical aspect of remote attestation is the establishment of a fresh shared key, since (1) previously used keys may have been compromised and (2) subsequent operations of the prover $\mathcal{P}$, like a software update, can be cryptographically bound to the device that succeeded in the attestation.

In contrast to purely software-based attestation, which are currently unable to establish a common shared key without assuming an existing uncompromised key at $\mathcal{P}$, our scheme can

Algorithm 1: High-level attestation protocol that pseudo-randomly interleaves hardware and software checksum functions `HwSum`, `SwSum`.

```
Input:
$r$: PRNG seed of the verifier's attestation challenge
$q$: CRP offset of the verifier's attestation challenge
Output:
$\sigma_k$: Final checksum value and attestation response
Temporary Variables:
$i$: Loop counter
$r_i$: PRNG state in iteration $i$
$x_i$: Input to HwSum in round $i$
$y_i$: Output from HwSum in round $i$
$a_i$: Input to SwSum in round $i$ (address of memory block to be measured)
$\sigma_i$: Intermediate overall checksum value in iteration $i$

AttestLocal($r, q$)
begin
    $(r_0, \sigma_0) \leftarrow$ PRNG($r$ ); for $i = 0$ to $k$ do
        $x_i \leftarrow q \oplus \sigma_{i-1}$
        $y_i \leftarrow$ HwChkSum($x_i$)
        $(a_i, r_i) \leftarrow$ PRNG($r_{i-1}, a_{i-1}, y_i$)
        $\sigma_i \leftarrow$ SwChkSum($\sigma_{i-1}, a_i, i, r_i$)
    end
    return $\sigma_k$
end
```

leverage the unclonability property of the hardware checksum `HwSum` () to generate shared secrets between prover and verifier. In fact, the final checksum value of our attestation protocol already includes a significant amount of data from the hardware checksum that is unpredictable to the adversary, as otherwise the hardware checksum and thus the complete attestation process could be simulated in software.

We can thus use the final checksum σ_k directly as a shared secret key: Instead of directly disclosing it to the verifier in the attestation response, we continue with explicit key confirmation, i.e., exchange proofs of knowledge of the common shared secret using cryptographic hashes of σ_k and additional random nonces from each party.

On Demand CRP Generation

As described in Section 6.5.1, our solution for non-simulatable PUFs requires the verifier to generate sets of Challenge-Response Pairs (CRPs) before deployment and to commit to the number of possible attestations during the life-time of the node. The approach can be problematic due to the required amount of storage at the verifier and, more importantly, it may be unsuitable for dynamic trust management, where the demand for a (possibly rather expensive) remote attestation procedure with $\mathcal{P}$ is determined by the individual behavior of $\mathcal{P}$, e.g., based on the correctness of its actions or reports.

An important extension of our attestation scheme is thus the generation of additional CRPs after deployment. While this would normally require either an exhaustive database of the exponentially large amount of CRPs per prover or a completely deterministic attestation routine with a fixed software state S, we can extend the attestation routine described in Algorithm 1 to maintain full flexibility: We modify the PUF challenge x with a reduction function $f()$, such that $x_i = (q \oplus f(\sigma_{i-1}))$. As a result, individual attestation procedures use a smaller CRP sub-space that is determined by the offset q chosen by the verifier and the size of the CRP sub-space, which is determined by $f()$: $2^{|f(\sigma)|}$.

To simulate the resulting attestation protocol for arbitrary software states S, the verifier must now commit to function $f()$ and offset q and store the resulting CRP sub-space. The required storage can be optimized by storing the PUF responses y_i for each offset q in a sorted list, with implicit list index x_i. Furthermore, we can shorten the bit length of the PUF responses y_i, e.g., by using only their least significant bit in the checksum computation.

To generate additional CRP sub-spaces on demand, the verifier instructs the attested prover to generate and transfer a new list of responses y_i for a given offset q. The transfer must be confidential, e.g., protected by the common shared secret established by the attestation scheme (see Section 6.5.2).

Cooperative Attestation

In wireless sensor networks, hop-to-hop communication combined with energy restrictions at the sensor nodes used for routing can impose high transmission delays and more importantly jitter. The time measurement of the attestation should therefore be delegated to the direct neighbors of the prover, as discussed in [YWZC07]. The measuring neighbor can record response time and forward it to the verifier in an authenticated fashion, together with the original response of the prover. Multiple neighbor nodes in the same broadcast zone can collaborate in this action and inform the base station about their measured time, resulting in a threshold-secure attestation scheme.

To attest the whole sensor network, the base station should start at its immediate neighbors and iteratively attest all nodes in the network. Note that the security of this approach also depends on the integrity nodes that the time measurement is delegated to, i.e., the time it takes the adversary to compromise a given set of nodes after attestation. However, the attack surface can be minimized by keeping nodes that are still to be used for delegated time measurements in a constrained mode, where not all software and hardware components are active.

6.6 Secure Routing

A survey of the related work (see Section 4.2.3) has identified two secure routing protocols that are designed specifically for a Heterogeneous Wireless Sensor Network (HSN): Two-Tier Secure Routing (TTSR) [DGXC07] and Resource Oriented Security Solution (ROSS) [CC07]. However, none of them meets security and functional requirements of TeSOS WSN. TTSR cannot be applied to the TeSOS network due to a weaker adversary model that assumes trusted cluster heads. ROSS does not require cluster heads to be trusted, but relies on reactive measures (detection and revocation of compromised nodes) rather than provides a tolerance against intrusion. Moreover, neither of these protocols support alternative routes to backup cluster heads, while such a functionality is required by TeSOS.

Generally, a hierarchical TeSOS WSN can be seen as consisting of two non-hierarchical layers. The first layer composed by BNs and BS provides inter-cluster communication. The second layer consisting of SNs and CHs is responsible for intra-cluster communication. In a normal operational mode no routing is required within the cluster, as SNs are able to communicate directly with their cluster heads (as depicted in Figure 6.7a). However, the network supports also a backup mode of operation which is applied in a case of failure or compromise of a main cluster head. In backup mode SNs can act as relays and forward data from other SNs (Figure 6.7b). To discover routes to backup cluster heads, intra-cluster routing is required.

The TeSOS WSN should provide a means to establish relationships between SNs and their main and backup cluster heads. Secure selection of backup cluster heads is not a straightforward task, as those are out of direct radio range of corresponding SNs. Typically, cluster formation and selection of cluster heads is a task of network clustering mechanisms. Existing works on network

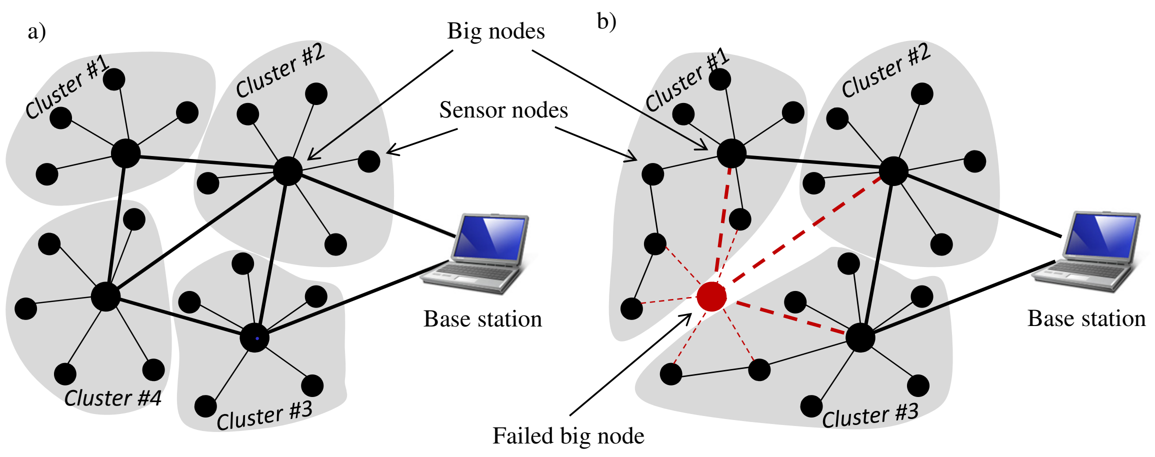


Figure 6.7: Operational modes of a clustered TeSOS network: (a) a normal mode; (b) a back-up mode

clustering do not address the problem of associating cluster members with backup cluster heads, thus a novel network clustering mechanism is required.

6.6.1 Network Clustering

In the following, we propose a novel network cluster representation which provides a means to establish relationships among SNs and their main and backup CHs. We represent a network as a set of *inner* and *exterior* clusters formed around cluster heads, as depicted in Figure 6.8. The inner cluster (depicted as grayed areas) includes only SNs located within a direct radio range from a cluster head. The radius of the inner cluster R_i has upper bound r, where r is a communication range of a small node.

The exterior cluster has radius $R_e \geq r$, thus, not all SNs are able to communicate a cluster head directly, but may reach it via hop-by-hop communication. Exterior clusters of respective cluster heads overlap, thus a single SN can be a member of several exterior clusters. For instance, in Figure 6.8, a small node d is a member of three exterior clusters formed around cluster heads A, B, C, and a member of an inner cluster of X. A membership in a single exterior cluster provides SN with a single route to a corresponding backup cluster head, while a membership in an inner cluster associates the node with its main cluster head. For instance, the node d has three routes to backup cluster heads A, B and C and is associated with a main cluster head X.

To form inner network clusters, SNs (randomly) select one of the available BNs in the direct radio range as a main cluster head. For forming exterior clusters, the location-based approach can be used, where a main cluster head supplies own location and location of its neighbors to members of inner clusters. SNs pick up (a defined number of) backup cluster heads among BNs that are neighbors of its main cluster head.

Analysis. The location-based approach for clustering requires location information to be trustworthy to defeat possible location manipulation-based attacks.

An alternative solution is to use flooding to advertise presence of backup cluster heads to SNs and hop count to limit flooded area. However, a hop counter included into a flooding message should be protected from manipulations, which is impossible to achieve in networks where nodes can be compromised. Indeed, a compromised node can arbitrary manipulate a hop counter of the forwarded message instead of decrementing it (as dictated by the protocol).

The location-based approach seems to be more feasible, but would require either certified BN location and ability to verify this information by SNs, or to trust to a main cluster head to accomplish such a verification. In TeSOS, the BN's location can be certified by a trusted BS,

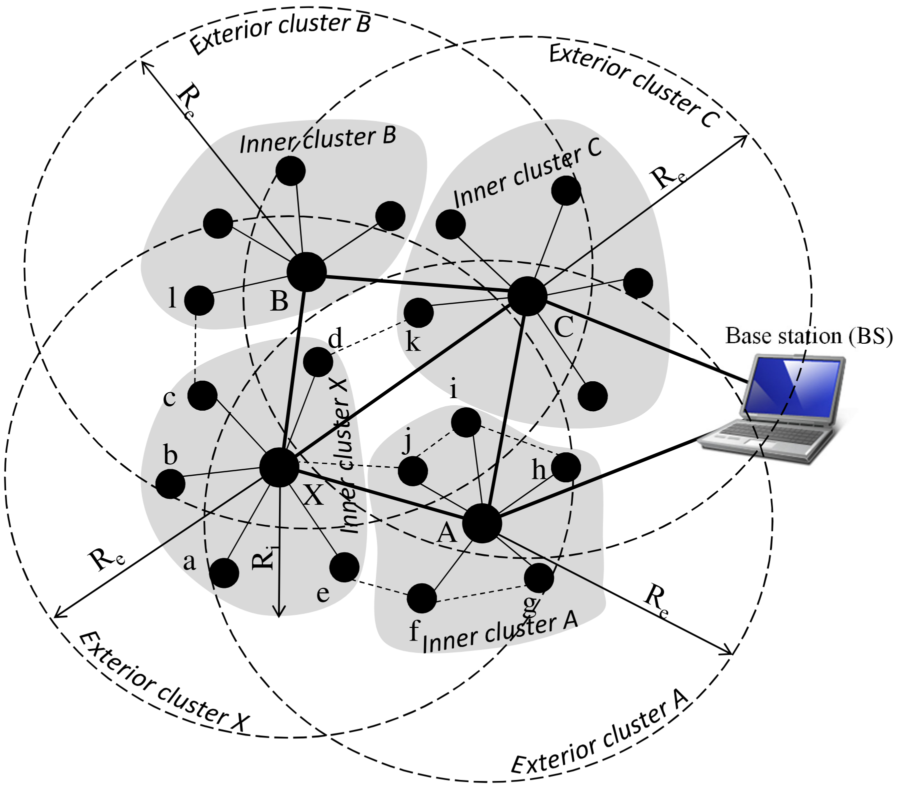


Figure 6.8: Network clustering

but neither are the BNs assumed to be trusted[1], nor can SNs perform signature verification.

A solution to eliminate the necessity to provide a trusted environment for network clustering is to enable SNs to verify location information themselves. When used with highly efficient signature verification this solution can be feasible even for resource constraint SNs.

An alternative approach which would not require trusted BNs and any Public Key Cryptography (PKC) operations from SNs is to run network clustering procedure in a secure environment, e.g., during network initialization, under assumption that an adversary cannot affect this phase (this assumption holds for the TeSOS network). However, following this approach would leave maintenance issues open, as addition of new BNs which might happen in operational phase would require re-clustering conducted in a secure environment.

6.6.2 Inter-cluster Secure Routing

Good candidates for inter-cluster routing in the TeSOS network are geographic (or location-based) protocols, as the TeSOS network is static and location aware. Geographic protocols are resilient to a number of routing attacks, such as as wormholes [HPJ03a], sinkholes [KW03], and HELLO flood [KW03], thus no additional mechanisms are required to defeat against them. We identified two location-based secure routing protocols which can be suitable for TeSOS: Resilient Geographic Routing (RGR) [AGKL05, KLAG06] and Dynamic Window Secured Implicit Geographic Forwarding (DWSIGF) [HIJM09].

These protocols follow the concept of lazy binding, where forwarding nodes select a next hop relay at the last moment before data is forwarded. To maintain lazy binding, nodes use Request-to-Send (RTS)/Clear-to-Send (CTS) handshake procedure.

RTS/CTS handshake is initiated by a source node broadcasting a RTS package. Typically,

[1]However, in Section 8.2.1 we propose an integrated solution of security mechanisms for TeSOS with trusted BNs

RTS includes the source and destination locations. All neighboring nodes located towards the destination (e.g., in the 60° sextant) replay with a unicast CTS message, pretending to be a next forwarding node. In RGR, a source node collects CTS replays from all neighbors before the next hop is selected, while DWSIGF collects responses during a collection window time.

Lazy binding increases fault tolerance, as a handshake reduces the chance that packets are forwarded to a failed or sleeping nodes, and also enables the use of newly arriving nodes. To eliminate selection of malicious nodes for forwarding, both protocols incorporate trust management systems that honor well-behaving nodes and punish suspicious ones. Moreover, protocols support principles of multi-path routing: k neighbors can be selected from the candidates for forwarding, thus enable redundancy to tolerate intrusions.

Analysis. As both protocols rely on location information to make forwarding decisions, this information must be trustworthy to ensure correct routing. RGR uses trusted nodes (anchors) to certify verified location information, while DWSIGF incorporates a metric for location consistency reported by the node into the trust management system. In TeSOS, location information can be certified by a base station, as BNs support PKC and can verify BS signature. Location information can be exchanged among BNs in a network initialization phase, as network nodes typically do not change their positions.

One challenging issue in adopting RGR and DWSIGF protocols to TeSOS WSN is the fact that RTS packages have to be sent as broadcasts, while authentication of broadcast messages in WSNs is not trivial. Although BNs support PKC, authenticating RTS messages with signatures would be extremely costly, as RTS messages are generated at each hop along the path. μ-TESLA broadcast authentication scheme [PSW⁺01] is a good solution for a small number of broadcast sources (as requires key chain synchronization among a sender and all receivers), but does not fit well in this particular case, where every node may need to send broadcasts. Moreover, μ-TESLA scheme provides delayed authentication, thus its application would result in a significant end-to-end delay accumulated at each hop.

An authentication scheme applied in SAPC [BLM07, BLMB07] can be considered as a candidate for this particular use case. In this scheme, the message is authenticated with a key from a one-way-hash chain (synchronized among a source node and all its neighbors), while the key is enclosed to the message. Although generally the scheme is vulnerable to a forgery attack (as was already discussed in Section 6.3), RTS message can be formatted in such a way that it does not contain any fields that can be forged. For instance, DWSIGF protocol proposes an option to omit location information in the RTS message. When location data is excluded, RTS message has only a message type and identity information of the sender. If a key chain is associated with a message type and a sender identity, an adversary cannot successfully forge this information.

When RTS node does not include sender and destination locations, the neighbors of a node can determine whether they are in the forwarding area by using knowledge of neighbor locations. Thus, those of them what are located in the area towards the destination should respond with CTS messages. When the data message is relayed to the selected node, it must contain the destination's location to enable subsequent routing.

6.6.3 Intra-cluster Secure Routing

Intra-cluster routing protocol is required to discover routes within exterior clusters. The most suitable protocol we identified is INtrusion-tolerant routing protocol for wireless SEnsor NetworkS (INSENS) [DHM03, DHM04, DHM06a].

INSENS protocol has two major phases: Route discovery and data forwarding. Route discovery collects information about the topology of the sensor network and sets up appropriate routing tables at each node by exchanging control messages. In a data forwarding phase messages are forwarded to their destination with accordance to routing tables stored on the nodes.

Route discovery is performed in three phases. In the first phase, the base station securely floods a *request message* to all nodes in the network. In the second phase, sensor nodes send their local topology information using a *feedback message* back to the base station. In the third phase, the base station validates the submitted by nodes local topology information and reconstructs a global network view, calculates multipath routing tables for each sensor node and unicasts those tables in a breadth-first manner to the respective nodes using a *routing update message*.

Analysis. Because all routes in INSENS are calculated by a trusted party (the BS), an adversary is limited in manipulating routing information. This strategy equalizes chances of compromised and non-compromised nodes to be selected for forwarding and rules out routing attacks where a compromised node makes itself attractive for forwarding by advertising a high quality route metric (e.g., a sinkhole attack). However, in the intra-cluster communication level of the TeSOS network, equivalents of the BS are cluster heads, that are not trusted. In case a backup cluster head gets compromised, it can manipulate all the routes within the exterior cluster. Though, we suppose it still can be applied in context of TeSOS, as compromised cluster heads can manipulate only a single backup route leading to itself, but cannot affect other backup routes. It makes no sense for the attacker to include more compromised nodes into the route, as the compromised cluster head is already inside. Still, a compromised CH can perform DoS attacks affecting all nodes of the exterior cluster. For instance, it can construct loops in the route, thus forcing SNs to continuously forward messages and waste energy.

Another problem of the INSENS protocol is hidden in a data forwarding phase: The protocol assumes broadcast transmissions for data delivery. All the nodes hearing a data message decide if they have to forward the message further based on a routing table. For broadcast authentication of data messages, the protocol relies on a MAC calculated with a symmetric key shared by a node and all its neighbors. This scheme has low resilience to a node compromise, because by compromising a single node the adversary will be able to forge messages of all its neighbors.

Taking into account mentioned above problems, INSENS is not a perfect candidate for usage in TeSOS network. A new protocol has to be designed that (i) does not entirely rely on cluster heads to calculate the path, but decision is taken cooperatively by the cluster head and cluster members, and (ii) does not require broadcast transmissions in a data forwarding phase. Currently, designing such a protocol is our work in progress.

6.7 Time Synchronization

For time synchronization we leverage the existing TinyReSync [SNW06] protocol implemented for TinyOS. It uses Secure Pairwise Synchronisation (SPS) for pairwise synchronization as described in [GvHS] but inserts time stamps at the MAC-layer as proposed in [GKS03]. In Secure Pairwise Synchronisation (SPS), the sender A issues an authenticated challenge at time T_1 containing a nonce N_A to the receiver B, who in turn responds with an authenticated message that contains the time T_2 when the challenge was received as well as when the response is issued (T_3) and the nonce N_A. A records the time when the response is received as T_4, computes the average end-to-end delay d and if it is within reasonable range, the clock offset o:

1. $A \rightarrow B$: A, B, N_A
2. $A \leftarrow B$: $B, A, T_2, T_3, N_A, MAC_{K_{AB}}(B, A, N_A, T_2, T_3)$
3. A: $d = ((T_2 - T_1) + (T_4 - T_3))/2, o = ((T_2 - T_1) - (T_4 - T_3))/2$

When leveraging a-priori knowledge on topology and physical network layout, plain SPS can be applied iteratively, similar to the Secure Transitive Multi-hop Synchronization (STM) protocol presented in [GvHS]: The base station starts with its immediate neighbor BN, which in turn synchronize the BN that are directly reachable from any of the synchronized nodes.

Once synchronized, each BN also synchronizes the respective SN that belong to its cluster. To synchronize time downwards in the WSN hierarchy, i.e., synchronize nodes further out from the base station according to their respective next closer hop, SPS must be extended such that the initializer of the message exchange is informed about the final clock offset. This can be done by either inserting an additional authenticated *sync* message from B to A to the exchange, or by appending a third message from A to B that informs B on the calculated clock offset o in an authenticated message. The first alternative reduces load on central parts in the network, by distributing computation load to the leaves. In contrast, the second alternative allows nodes more close to the base station to collect information about network drift of their neighbors.

7 Operating Systems for Sensor Nodes (M5)

Several embedded operating systems for sensors have been developed in the last few years. Examples are operating systems such as TinyOS [HSW+00a], MANTIS [BCD+05a], Contiki [DGV04], or SOS [HKS+05]. However, none of them protects the operating system from the applications due to the fact that many micro controllers do not provide hardware support for security features[1]. Therefore, it is difficult to create reliable sensor network software that runs safely on these operating systems.

In this chapter, the most common operating systems for embedded systems and Wireless Sensor Networks (WSNs) will be analyzed and compared. The following Section 7.1 discusses the criteria used for the analysis and comparison, followed by a detailed analysis of the different embedded operating systems in Section 7.2 to Section 7.7. Section 7.8 summarizes this chapter and suggests embedded operating systems to be used with TeSOS.

7.1 Criteria of Operating System Evaluation

To cover the huge amount of different operating systems in the world of embedded devices, which are also used in the area of WSNs, detailed criteria have to be defined. Each criterion handles its own district inside the operating system, thus each of these criteria is specified and defined in the following section.

7.1.1 Process Management

One of the most central aspects in any type of operating system is the process, an abstraction of a running program. These processes have to be handled in a proper way, since very often strong relationship between different processes exists, causing dependencies that have to be solved during process life-time. If only one Central Processing Unit (CPU) is available, the system has to make a choice which process to handle next. Therefore, also the scheduler is of importance since a good scheduler can make a big difference in perceiving performance and user satisfaction. In the following, the most important scheduler types are briefly introduced:

- *First-Come First-Served:* This type of scheduling algorithm is the simplest method to manage non-preemptive processes. With this algorithm there is a simple queue of processes that are assigned to the CPU and processed in sequence. New processes are appended at the end of the queue. If a process is blocked it will also be appended at the end of the queue. The biggest advantages of this type of algorithm is that it is very easy to understand and implement. Unfortunately, first-come first-served also has disadvantages, for example, if a process is combined with input-/output processes, this could block the whole system.

- *Shortest Job First:* When several jobs with an equal importance are present, this scheduling algorithm picks the shortest job first. However, it is provably optimal, it still could cause

[1]A incomplete list of hardware-based security features has been presented in Section 5.1.4

the problems of blocking systems if a very short job arrives, but is not yet ready to be processed. Resulting other jobs will be processed before, the algorithm does not behave optimal any more.

- *Short Remaining Time Next:* This algorithm is a preemptive version of shortest job first. Here, the scheduler always chooses the process whose remaining run time is the shortest, which, of course, only works, if the runtime is known in advance.
- *Three-Level Scheduling:* At the beginning of this scheduling-scheme, all jobs are stored inside an Input Queue, which is in most of the cases a linked list. After this queue, the Admission Scheduler decides which job is going to be delivered to the system (the memory). It is also possible that the Admission Scheduler prefers some other jobs inside its own list. Being delivered to the memory, the Memory Scheduler decides which jobs are swapped to the other external storage. Since this swapping is often very expensive, very short jobs are normally not swapped to the external memory. Last step in the Three-Level Scheduling is the CPU Scheduler, which picks the jobs from the Memory Scheduler and processes them until finished, regarding aspects, such as process-imminentness, -size, and -time the process had before.
- *Round-Robin Scheduling:* Using Round-Robin Scheduling, each process is assigned a predefined time interval. If the interval is not big enough, the process is transfered to the end of the internal process list and the next process in the list gets access to CPU time.
- *Priority Scheduling:* In comparison to e.g., Round-Robin Scheduling, Priority Scheduling assigns another indicator to current processes of the system - priority. For example, some nuclear plant real-time process gets a higher priority than a process delivering emails to the user. Also it is possible to prevent processes from running indefinitely, which is done by the ability to lower the priority of currently running processes.
- *Multiple Queues:* To reduce swapping, the Multiple Queues algorithm increases the assigned CPU time every time a process is currently running. This results in a very fast job processing for small jobs. Bigger processes would run less and less frequently, saving the CPU for short, interactive processes.
- *Shortest Process Next:* Since Shortest Job First produces the minimum average response time for batch system, a similar system in interactive systems would be necessary. Shortest Process Next handles each process in interactive systems as a pattern of waiting for commands and executing commands. Execution of the commands is handled as job, whose execution time is estimated in this scheduling algorithm. Followed by this estimation, the Jobs are ordered by their execution time, which produces an approximately minimal average runtime.
- *Guaranteed Scheduling:* In comparison to other scheduling algorithms, Guaranteed Scheduling ensures that each process get exactly the same amount of CPU time. For example, there are 10 processes, every process gets a guaranteed CPU-time of 1/10. If one of the 10 processes uses more than 1/10 of its time, it is automatically lowered in its priority.
- *Lottery Scheduling:* Using the Lottery Scheduling algorithm, each process gets tickets to various system resources, such as CPU time. Each scheduling decision that has to be made, one ticket is chosen randomly. A process with more lottery tickets has a bigger chance to get the system resource than other processes with less tickets.
- *Fair-Share Scheduling:* Compared to all other scheduling algorithms, the Fair-Share Scheduling algorithm also integrates the owner of processes to the internal scheduling. For example, user 1 produces 5 processes, user 2 produces only 1 process. If both users are scheduled with

50 percent of the CPU time, process number 6 (the one from user 2) would be scheduled every 2nd time a process runs.

Using the above observed scheduling algorithms in Real-Time systems is often very difficult or even not possible. An autopilot of an aircraft, or some monitoring in an intensive-care unit in hospitals need to have their CPU time at a predefined level. In general, real-time systems are divided into (i) hard real time systems with its absolute deadlines that must be met and (ii) soft real time systems, meaning that missing an occasional deadline is undesirable, but still tolerable. Furthermore, the events in a real-time system can be categorized as periodic or aperiodic, regarding their predictability of their system behavior. The job of the scheduler is to schedule all processes in such a way that all deadlines are met.

Regarding thread scheduling, two levels of parallelism have to be distinguished: processes and threads. Scheduling in such systems depends on whether kernel-level threads or user-level threads are present in the system. With user-level threads, due to the fact that the kernel is not aware of the existence of threads, it always gives each process a specific CPU-time to run. The thread scheduler within this process decides which thread to run next. In comparison to user-level threads, with kernel-level threads the kernel picks a particular thread to run. It does not have to take care about, which process the thread belongs to. The major difference between both approaches is the performance. Processing a thread switch with user-level threads takes only a few machine instructions, in contrast a thread switch with kernel-threads requires a full context switch, changing the memory map and invalidating the cache. Also, it has to be mentioned that using kernel-threads, a block of some input or output device does not suspend the entire process as it does with user-level threads.

7.1.2 Resource Management

Managing resources of computer systems that can only be used by one process at a time, such as printers, tape drives, and so on, can cause the system to enter a situation where several processes block each other, called deadlock. For example, having two processes using the same file system table slot will invariably lead to a corrupted file system. In consequence, all operating systems have the ability to grant a single process access to a specific system resource.

Categorizing the resources, there are preemptable resources, such as the specific amount of memory and non-preemptable resources, such as if a process begins to burn a DVD and the DVD recorder is suddenly taken away. In general, deadlocks are generated by non-preemptable resources, preemptive deadlocks can usually be resolved by reallocating resources from one process to another. Four conditions must be present for a deadlock to occur:

- *Mutual exclusion condition:* Either a resource is available, or it is assigned to exactly one process.
- *Hold and wait condition:* A Process that allocates a specific resource before, is still able to request for additional resources.
- *No preemption condition:* Each resource that has been assigned to a specific process can not be taken away from the process without the process releases it on its own.
- *Circular wait condition:* In a chain of processes waiting for resources, other processes of the chain hold the requested resources.

Deadlocks can be detected, avoided, and resolved by algorithms, such as the Banker's Algorithm[2] or by preventing specific conditions as mentioned before.

Since memory is a very rare resource, especially in embedded systems, special memory management mechanisms must be present to share the available amount of memory in a proper way.

[2]Described in http://www.isi.edu/~faber/cs402/notes/lecture9.html

The part of the operating system that manages the memory is called memory manager. Its job is to keep track of which parts of memory are in use and which parts are not in use, if needed to allocate some memory to specific processes and to manage swapping between different types of memory, such as main memory, flash, or special caches of embedded hardware. Commonly used memory management algorithms include:

- *Monoprogramming without Swapping or Paging:* Only one program is running at a time in parallel to the operating system. Organized in this way, only one process can run at a time.
- *Multiprogramming with Fixed Partitions:* Since today almost all embedded systems run several processes in parallel, available memory has to be available for all processes, ideally in parallel. The easiest way to achieve this is by dividing the available memory into partitions, where each process gets access to the first partition big enough to hold it. Different algorithms are available to solve problems, such as *relocation* and *protection*.
- *Swapping:* Using fixed partitions in a batch system is very simple and effective. In most cases time sharing is used to keep the CPU busy all the time. Therefore, a different technology is needed if not enough memory for all active processes is available. The simplest way to solve this issue is using a technology called swapping, where a process is running for a while and than its state is put back on persistent memory.
- *Paging:* An even more complex strategy to the issue targeted by swapping is called paging, which allows programs to run even when they are only partially in main memory.

7.1.3 Protection

Introducing the protection of embedded operating system, it has to be defined what an operating system should be capable to protect. In general, the following aspects need to be taken into consideration by the operating system software:

- *Direct access to hardware:* Since the operating system is the instance that has to be fully trusted, it has to control the access to the hardware of the embedded device. It defines, for instance, which process is granted access to the input and output devices, or which process is prioritized and which not. Sharing the CPU-time and protecting the taken decisions is a very important aspect of each operating system.
- *Memory protection within several processes:* Since there are usually several processes running in parallel in common embedded systems, the available memory has to be managed to guarantee that no process gets access to the memory of another process. In hardware this is mostly realized by using an Memory Management Unit (MMU), however, not all hardware platforms are equipped with an MMU.

One of the protection mechanisms in operating systems is called *protection rings*. Here, each process is running at a fixed level beginning with *"ring 0"*, which represents the kernel itself, up to *"ring 3"*, which represents the user programs. Each *"ring"* has its own permissions to grant or to restrict access to hardware or memory.

Another common protection mechanism is called access matrix. Here, a system wire matrix is presented, which consists of the user rights per user and hardware. Each time a process owned by a defined user tries to get access to parts of the system, the operating system checks its access matrix, whether the process or user is authorized. More general but especially addressing current problems of data leakage, the Bell-LaPadula security model, presented in 1973, was developed at the government funded Mitre Corporation, in order to realize a new security model for military computer usage. The goal of Bell-LaPadula was to grant data leakage prevention inside an operating system. The basic concept is that an unclassified user should not be able to read classified information, while a classified user should not be able to write classified information into an unclassified part of the system. Therefore, different security levels are defined, where each user or process belongs to. Very important in this scenario is a proper enforcement of given rules inside the underlying system, thus it can be mathematically proven to prevent information at any given security level from flowing to a "lower" security level. Typically security levels are called “secret“, “unclassified“ or “classified“, rules could be either reading or writing, defining if reading or writing is allowed or not. A first realization of such a system inside embedded sensor nodes and WSNs is presented in [BGHL10].

7.1.4 Power Management

Since power management in sensor nodes is one of the most important aspects, the operating system is playing an important role here. In general, there are two approaches to reduce energy consumption. The first is to turn off parts of the sensor (input-/output devices, disconnect parts of the memory from power-supply, disable radio devices, and so on). The second approach is to optimize the application programs in such a way that the CPU consumes less time to achieve equal results. Important aspects of power management are:

- *CPU:* Common operating systems support features, such as the reduction of the speed of the CPU. A reduction of the speed directly reduces the power consumption of the overall system, which of course increases the necessary time to compute a given problem or to complete its processes. Often the used CPUs supports special sleep modes that can be addressed by the operating system directly.
- *Memory:* In the world of embedded systems, it is sometimes possible to disable parts of the system memory. Disconnecting these areas from power saves energy and therefore increases the life-time of sensor nodes equipped with such features.
- *I/O:* Similar to the memory, it is sometimes possible to disconnect several input- and output devices, which also decreases the power-consumption of the embedded system.
- *Multi-Core support:* Upcoming systems are equipped with more and more performance. The general trend is to increase the amount of cores inside the CPUs. As long as the used operating system and its running processes are capable to divide a computing problem into several threads, this feature increases dramatically the overall performance of the system. However, it commonly occurs that the current running program does not need (or is not able) to be computed within multiple CPU cores. In this case, disconnecting the additional cores will save a lot of energy. This behavior must also be controlled by the operating system.

7.1.5 External Factors

In addition to criteria, such as power management, protection, or resource management, some boundary conditions could also be of interest. Accordingly, this analysis will also mention the following aspects:

- *Circulation:* If a WSN is deployed, one of the main aspects is that there must be support for hard- and software over a predefined period. It can be expected that operating systems with a wide spread circulation do have a bigger community and therefore circulation helps to guarantee a better support of these systems.
- *Assurance:* In security-critical, governmental, or company-based scenarios, a certification of the operating system is becoming more and more important, thus this analysis analyses whether the system is certified according to some criteria.

- *Actuality:* Similar to circulation, a high actuality of an operating system is very important. This document tries to mention how frequent there are system updates.
- *Country of Origin:* For special security or financial reasons, it could be of interest who is developing the operating system and what the country of origin is.
- *Programming Language:* Not all operating systems use (or are based on) the programming language C. Possible other languages include nesC, C++, Basic, or languages based on assembler.

7.1.6 Total Cost of Ownership

Estimating the price of the realization of a specific project is one of the most important criteria. Realizing projects for WSNs requires appropriate hardware and later costs for soft- and hardware updates to keep the sensor nodes running during project lifetime. Also important cost aspects are values, such as the software development and the hardware development. Since the operating system is one of the most important parts of the overall system, this document also figures out the licensing model. In this context the Total cost of ownership (TCO) will be mentioned, since the TCO is a financial estimation, having the purpose to help determining direct and indirect costs of a product or system. Also of interest is the licensing model of each operating system, since it could be possible that modifications of the software needs to be done during project lifetime.

7.1.7 Manageability

Realizing projects with embedded systems often requires a deep knowledge of the underlying hardware system, thus the realization of WSNs often requires an adaptation of the used software to achieve a complete system, running in the proposed way. Knowledge of the hardware is one thing, understanding the internals of the operating system another. To be able to modify the operating system, one needs two important requirements:

- *Documentation of the operating system:* The more complete and structured the documentation of an possibly highly complex software system is, the more easy is every possible software adaptation. Thus, a well documentation of the operating system significantly supports the development process.
- *Licensing model:* Even if the documentation of the operating system is well written, a fitting licensing model is still necessary to be allowed to modify hardware drivers or other components within the context of the operating system. Modifications of the operating system are necessary, if it does not support the features that should be included in ones WSN project. Also, the pricing structure of the used software could be of interest.

An additional criteria when analyzing an operating system is the usability. “How much time do software developers need to get familiar with the system?”, and “Is there any software development environment available?” are questions to be answered during analysis and will be addressed in this document.

7.1.8 Performance-Analysis

Although the performance of operating systems is an important aspect, the realization of a full performance analysis is not feasible within this study. Several hardware devices, each capable to run all operating systems to be compared, would be necessary to achieve sufficient results. Influencing aspects of the system performance would be mainly the used scheduler and the performance of caching and communication between different processes. Since no hardware measurement is done in this study, no detailed performance analysis can be done.

7.1.9 Using the operating system

This section will introduce some basic aspects on how to install and use the analyzed operating system. If possible, basic examples will be given.

7.1.10 Applicability for TeSOS technologies

This section analyses the applicability of results of Chapter 6 to be used in each embedded operating system. It is mentioned whether it would be possible to modify the operating system to implement the proposed features.

7.2 eCos

The embedded operating system eCos (embedded configurable operating system) is a real-time embedded kernel providing thread scheduling, synchronization, timer, and communication primitives. Currently, the most recent version of eCos is version 3.0. The operating system handles hardware resources such as interrupts, exceptions, memory and caches. On the application level, eCos supports the μITRON and the POSIX Application Programming Interfaces (APIs), giving the possibility to run POSIX-conform software on top of this embedded kernel. eCos is implemented in C/C++, includes a thread-safe ISO standard C library, and a math library. eCos was designed to be able to run on hardware platforms that are not equipped with enough memory to run embedded Linux that needs about 2 Megabytes of Random Access Memory (RAM). The eCos Package Administration Tool can be used to add or remove different software packages from the component repository. The eCos Configuration Tool can include or exclude these packages from the configuration being built.

In comparison to other operating systems, the eCos kernel itself is an optional package not required to build very simple applications. Moreover, memory allocation is handled by a fully separated package. All device drivers are represented as attachable packages. The above mentioned configuration tool is then used to integrate all required packages into one application. eCos supports multiple hardware platforms, including

- ARM,
- CalmRISC,
- FR-V,
- Hitachi H8,
- Motorola 68000,
- Matsushita AM3x,
- MIPS,
- NEC V8xx,
- Nios II,
- PowerPC,
- SPARC, and
- SuperH.

For some of these platform, eCos offers limited Synchronous Multiprocessing (SMP) support.

In addition to the operating system itself, the software package is distributed with RedBoot. The RedBoot ROM monitor is an application that uses the eCos Hardware Abstraction Layer (HAL) for additional portability. RedBoot provides serial-based and ethernet-based booting and debug services during development. In addition, eCos offers drivers for

- file system,
- FLASH,
- Serial Peripheral Interface Bus (SPI),
- Inter-Integrated Circuit (I^2C),

- Controller Area Network (CAN),
- Analog-to-Digital Converter (ADC), and
- frame buffer.

For networking, a TCP/IP stack is available. It is even possible to use IPSEC or PPP, if the FreeBSD network stack is used. A simple embedded HTTP server, aimed to remote control applications on the embedded device, an File Transfer Protocol (FTP) client, sn Simple Network Time Protocol (SNTP) client, and an implementation of the POSIX Cyclic Redundancy Check (CRC) calculation are also provided. Features, such as Universal Serial Bus (USB)-slave, USB-ethernet, USB-serial, and MultiMediaCard (MMC)-support with an upper limit of 4 GB are also provided.

7.2.1 Process- and Resource Management

Using eCos, it is possible to write single-threaded applications without an eCos kernel. One example of such a single-thread application is the above mentioned RedBoot. In general, single-thread applications are managed by only one central loop that handles continually checking of all device drivers and possible interaction with input- and output devices. Of course, each calculation generates additionally delay for any interaction with the nodes I/O. If the requirements are more complicated, a multi-threaded application can be build by using the eCos kernel. The eCos kernel is also required for more complex software packages such as the TCP/IP stack. When the kernel package is used, the driver functions directly map to the equivalent kernel functions. Without a kernel, the common HAL package implements the driver API directly. Using the kernel functionality, two different possibilities are given:

- *Using the kernel's own C-API:* eCos-specific functions such as cyg_thread_create() and cyg_mutex_lock() can directly be used from application code or by other packages.

- *Using a standard API:* In addition to the kernel API, it is possible to use existing APIs, such as POSIX threads or μITRON, allowing application code to call standard functions, such as pthread_create(). Using these packages of course improves the possibility to reuse applications in different environments.

Since there are some differences between the semantics of API calls (e.g., to be strict μITRON compliant it is required that kernel time-slicing has been disabled) there are two important limitations. On the one hand, it is not always possible to combine two different compatibility packages into the same eCos configuration due to conflicting requirements of the underlying eCos kernel. On the other hand, the kernel's own C API is less detailed defined, because, for example, the command cyg_mutex_lock() will always try to lock a defined mutex, hence various other configuration options inside the kernel determine the behavior when the mutex is already locked, leading to the possibility of a priority inversion.

eCos offers different types of events that can be monitored. Examples of these events are:

- scheduler events,
- thread operations,
- interrupts,
- mutex operations,
- binary semaphore operations,
- counting semaphore operations, or
- clock ticks and interrupts.

If multi-threading is necessary, the kernel can be configured with one of two possible schedulers, described in the following:

- *Bitmap Scheduler:* The bitmap scheduler is more efficient, but has a number of limitations. The scheduler only allows one thread per priority level, thus amount of usable threads depends on the maximum priority-level.

- *Multi-Level Queue (MLQ) scheduler:* Almost all eCos based systems use the MLQ scheduler, which is described in Section 7.1. It allows multiple threads to run at the same priority-level. Therefore, the number of threads of an eCos application is only limited by the amount of available memory. However, some operations, such as finding the highest priority runnable thread, of the MLQ scheduler are more expensive due to its higher complexity. The MLQ scheduler also supports time-slicing.

Both schedulers, the bitmap and the MLQ scheduler, use a one-byte-based priority to define which thread should currently run. The number of priority levels is configurable via the option CYGNUM_KERNEL_SCHED_PRIORITIES. Typical, embedded system have up to 32 priority levels, therefore priorities will be in the range from 0 to 31, where a low number indicates a high priority. The priority 0 thread will usually be the idle thread of an eCos application. The kernel automatically decreases the priority-level of blocked threads.

The used scheduler is capable to handle event types, such as:

- scheduler lock,
- scheduler unlock,
- rescheduling or
- time-slicing.

The default behavior of an eCos system is last-in-first-out queuing, i.e., if multiple threads are waiting on a semaphore and an event is posted, the thread that gets access to CPU-time next is the last one that called cyg_semaphore_wait(). This implies that the thread with the highest priority is not always woken up. In contrast, if the bitmap scheduler is enabled, priority queuing is automatically enforced. However, some kernel functionality is only supported with the multi-level queue scheduler, in contrast to the bitmap scheduler. This includes support for SMP systems and protection against priority inversion using either mutex priority ceilings or priority inheritance.

The eCos HAL provides a device driver API, containing some of the mentioned synchronization methods. This API, e.g., allows an interrupt handler to signal events to higher-level code. Moreover, synchronization between threads within eCos is provided using the following primitives.

- *Mutexes:* Mutexes serve different purposes from the other primitives. It allows multiple threads to share a resource safely: A thread first locks a mutex, manipulates the shared resource, and then unlocks the mutex again.

- *Condition Variables:* These variables are used by mutexes so that it is possible to wake up threads by other threads. When a thread waits on a condition variable, the mutex is released before waiting. It releases the mutex before waiting, and when it wakes up it reacquires it before proceeding. Since these operations are atomic, race conditions are excluded.

- *Counting Semaphores:* A counting semaphore is used to indicate the occurrence of a particular event.

- *Mail Boxes:* If an event occurred, mail boxes are allowing one data item to be exchanged per event. Typically, this data item is a pointer to a data structure. Since this additional data-storage needs memory on his own, there exists a predefined maximum number of mail boxes, since the amount of memory is limited very strictly. If no more memory is available, the thread will be blocked until enough new memory is available.

- *Event Flags:* Event flags can be used to wait on a number of different events, and to signal that one or several of these events have occurred. Every occurred event is flagged with some corresponding bits in a bit-mask.

Using these synchronization primitives, the interrupt handler can signal a condition variable, post to a semaphore, or use one of the other primitives. The thread would perform a single wait operation on the same primitive. This would not consume any CPU cycles until the input-/output event had occurred, and when the event does occur, the thread can start running again immediately.

The eCos kernel uses a two-level approach to implement interrupt handling. Every interrupt vector is associated with an Interrupt Service Routine (ISR) that will be privileged regarding priority and CPU time such that it can service the hardware as fast as possible. This ISR can only make very few calls from the kernel, and it is impossible to make system calls that can wake up other threads. If it is detected that an input-/output operation has been finished and a corresponding thread should wake up, the ISR can cause the associated Deferred Service Routine (DSR) to run, which is equipped with more rights and is allowed to make additional kernel calls, e.g., signaling a condition variable or posting to a semaphore. In some cases, it is possible for threads to disable interrupts for a few instructions. Similar to disable interrupts, the kernel has a scheduler lock. Kernel functions, such as cyg_mutex_lock and cyg_semaphore_post(), rise the scheduler lock, manipulate the kernel data structures, and then release the scheduler lock again. If an interrupt results in a DSR being requested and the scheduler is currently locked, the DSR remains pending. When the scheduler lock is released, any pending DSRs will continue running, possibly posting events to other synchronization primitives, causing other higher priority threads to be woken up.

Since only certain calls are allowed from inside each context, eCos defines the following contexts:

- *Initialization:* During startup of the system, eCos sets up the embedded hardware and initializes the C++ static constructors. During initialization, interrupts are disabled and the scheduler is locked. The final operation of the initialization is a call named cyg_scheduler_start() which enables interrupts, unlocks the scheduler, and delivers control to the thread with the highest priority. It is also possible to run some applications before the scheduler is started. Thus it is possible to run, e.g., the constructors of a written C++ application or a special optional C function called cyg_user_start().

- *Thread:* The Initialization process is ending with the call of cyg_scheduler_start(). In the context of threads, almost all kernel functions are available.

- *Interrupt Service Routine:* If the processor receives an external interrupt, control is transferred from the current thread. The system will then switch to the appropriate interrupt service routine which can be provided by a HAL package, a device driver, or by the application itself. Being in the context of the ISR also means that most of the kernel functions calls are not usable, including synchronization primitives, such as semaphores to indicate an event.

- *Deferred Service Routine:* After leaving the ISR context, the system can enter the deferred service routine context for e.g., running alarm functions. The DSR context only allows a few kernel functions, even if there are more functions possible, compared with the ISR context.

eCos follows a common error handling strategy: In principle, error handling is possible to ensure that each function is used correctly. If an error is detected, a suitable error code is returned. For example, the POSIX function pthread_mutex_lock() can return various error codes including EINVAL and EDEADLK. Several problems may occur if applications implement error handling routines:

- Error handling during mutex lock or any kernel function requires CPU resources not to be underestimated.
- It is not unusual that functions return error codes which are not handled by the calling function, since the programmers often expect that the called function works as expected.
- Each caller of a function has to check for possible errors returned by the called function. This also costs CPU cycles and increases the code size of the application.
- It can happen that an error occurs where there is no way to handle it, e.g., EINVAL. The only thing an application can do in this case is to abort the application.

In contrast to common error handling, the eCos now does the following: Functions, such as cyg_mutex_lock() will not return an error code. Instead, eCos functions contain a couple of assertions which can be enabled or disabled. Normally, these assertions are left enabled during the development phase and a lot of kernel functions will perform parameter checks and other system consistency checks. If a problem is detected, an assertion will be reported and the application will be terminated, allowing developers to check for possible errors. If the application is released, these assertions are disabled during compile-time. In general, this is similar to a debug-flag of some common desktop applications where additional debug-operations are included with some compiler-flag.

An error condition of several APIs functions is lack of memory. Some kernel functions allocate some memory dynamically for some thread stack or other per-thread data. If now the used hardware is not equipped with enough memory or the application contains a memory leak, the function call would fail. The eCos kernel now avoids such problems by never performing dynamic memory allocations. Instead, it is the responsibility of the application code to provide all memory required for kernel data structures. A lot of applications handle this with defining data structures statically and not dynamically. Problems, such as memory fragmentation or leaks, cannot occur if all data is allocated statically.

Since eCos partly supports SMP, some adaptations are required, e.g., the scheduling has to be different since now threads have to be divided into several CPUs leading to additional race conditions. During the initialization phase, the system behaves as if only one CPU is present. In principle, the used scheduler lock is converted into a nestable spin-lock, achieved by adding a spin-lock and a CPU id to the original counter. If a thread causes a scheduler lock and the CPU id is equal to the current CPU then it can increment the internal counter and continue. If the id does not match, the CPU must spin on the spin-lock, after which it may increment the corresponding counter and store its own identity number in the CPU id. Releasing the lock again is possible through decrementing the counter. If the counter reaches zero, the spin-lock is cleared. In the current version of eCos, SMP is only supported by the above mentioned Multi-Level Queue scheduler. Therefore, the scheduler now divides its threads to run the threads with the highest priority on all available CPUs.

7.2.2 Protection

eCos does not support any additional protection models, nor restricts the direct access to hardware, or secure mechanisms to grand or deny access to the memory (except by the already described resource-management). There are no protection rings or any support for trusted anchors as introduced in Section 5.1.4.

7.2.3 Power Management

Although the eCos operating system provides a package responsible for power management, functionality to save energy is very limited. The package does not support any reduction of CPU speed or any other power-saving method provided by the hardware. In comparison, eCos provides a HAL, offering further implementations to easily support additional power management functionalities of the underlying hardware. A function or power management policy that forces the system to enter any low-power modes is also not available, however, the provided package builds a base framework to implement these features by your own. Power modes defined in the power management packages are:

- *Active:* The system is running at maximum speed and fully operational. It can be expected that power consumption is at the maximum.
- *Idle:* If there is almost no or only a little activity over a predefined short period of time, the system would be in the idle state. This causes a reduction of the CPU speed or disabling some of the input- or output devices. It has to be mentioned that due to the fact the idle thread is always running, it has to be defined when to enter the idle state. After some interrupt occurs, the idle mode can easily be left again.
- *Sleep:* Also to be defined by some power management rules, if the system is in idle mode for a predefined period of time, the system will go into the sleep mode. In this state it is expected to shut down almost all power-consuming devices of the hardware.
- *Off:* Shutting down the system means that the power consumption is minimized. All attached hardware is disabled. Possibly it is necessary to do additional actions to wake up the system again.

The main function of the power management package is to control all available power controllers of the hardware platform. These power controllers could be controllers for LCD displays or CPUs. It often happens that a hardware platform implements its own power controllers as part of the relevant device driver package, it also can happen that there is no hardware support for any power management at all. The eCos documentation gives some short exampled how to implement a basic power controller.

7.2.4 External Factors

eCosCentric Limited, a 2002 founded commercial provider of eCos products and services located in the United Kingdom, affirms that in the fragmented embedded operating systems market eCos is one of the major players, with global market usage of around 5-6 percent according to multiple surveys including CMP's Embedded Study 2007, and EDC's Embedded Development Survey 2007[3]. Famous products using eCos are e.g., Playstation 3 (used to provide the Playstation's Wi-Fi support based on the Marvell 88W8580 WLAN chip) or Samsung's LCD HDTVs, including the latest "M" and "F" top-of-the-line ranges of 32-70 inch sets. The TVs feature multi-media playback via USB2 from cameras, mp3 players, and flash drives. A detailed list of products/projects using eCos can be found at `http://www.ecoscentric.com/ecos/examples.shtml`. The used programming language for eCos applications is C/C++.

Beginning around the year 1997, eCos continually receives updates on supported hardware or additional packages, such as the PPP stack in April 2004. The current eCos version 3.0 was released in March 2009. However, a lot of contributions to the project were implemented from single persons, while the big releases are contributed by eCosCentric. Parts of the system also are contributed by RedHat, a leading open source provider located in USA and Canada. Except of a small number of packages, the copyright of the eCos public source repository is owned by the Free Software Foundation (FSF), thus the origin country of eCos is the same as the FSF. The Free Software Foundation Inc. is located in Bosten, USA.

At the moment, no certification of eCos is available.

[3]`http://www.researchandmarkets.com/reports/c90477`

7.2.5 Total Cost of Ownership

Since eCos is licensed under the terms of a modified version of the GPL, there are no direct costs on purchasing the operating system. Indirect costs can hardly be estimated since this heavily depends on the scenario of the project eCos is used for. Support of eCos is possible in two ways:

- *Mailing List:* Free and open to everyone is the public mailing list of eCos. Core developers read and post there.
- *Commercial:* It is possible to buy commercial support for eCos and/or RedBoot from a number of different vendors. This support includes, e.g., support for tool chain, new features of drivers, or porting to a new platform.

7.2.6 Manageability

eCos is one of the open source kernels that has been published under the terms of the GPL. For an easy start of development with eCos, it offers an easy graphical configuration tool to help users to configure and build a custom version of the operating system. Two versions of the configuration tool are available. While version 1 still provides more features, it can be expected that version 1 will be fully replaced by version 2. Version 2 is a cross-platform application using the wxWindows toolkit (http://www.wxwindows.org). The tool uses the GTK+ widget set under Linux, and the WIN32 API on Windows.

The source code of the configuration tool is also available using CVS and an anonymous account. In addition to source code, binary packages are also available. A user-guide of eCos is also online available at `http://ecos.sourceware.org/docs-latest/user-guide/ecos-user-guide.html`.

From the authors perspective, eCos is well documented including examples describing how to port the the operating system or its HAL to new platforms and how to develop new drivers. Based on the fact it is licensed under the terms of the GPL, eCos offers a wide variety of additional developments and projects in the area of embedded systems.

7.2.7 Using the operating system

Following the instructions at `http://ecos.sourceware.org/getstart.html`, additional packages might be necessary to be installed. At least *libstdc++ v3* must be present. After downloading the installation tool using:

```
wget --passive-ftp ftp://ecos.sourceware.org/pub/ecos/ecos-install.tcl
```

eCos and a corresponding tool chain can be installed with the command:

```
sh ecos-install.tcl
```

After installation, a first eCos application could be build using:

```
ecosconfig new <target_name> redboot
ecosconfig import $(ECOS_REPOSITORY)/hal/<architecture>/<platform>/_
<version>/misc/redboot_ROM.ecm
ecosconfig tree
make
```

This will produce the RedBoot ROM monitor mentioned before to ensure eCos is running in a very simple version. Connecting a device with a serial interface will then create the following output:

```
RedBoot(tm) bootstrap and debug environment [ROMRAM]
Non-certified release, version UNKNOWN - built 15:42:24, Mar 14 2002

Platform: <PLATFORM> (<ARCHITECTURE> <VARIANT>)
Copyright (C) 2000, 2001, 2002, Free Software Foundation, Inc.

RAM: 0x00000000-0x01000000, 0x000293e8-0x00ed1000 available
FLASH: 0x24000000 - 0x26000000, 256 blocks of 0x00020000 bytes each.
RedBoot>
```

If this output is visible, the initialization phase and the serial connection is working properly.

7.3 Contiki

Another open source minimal operating system is Contiki. The current version is 2.5 release candidate 1, published in November 2010. Contiki is a multi-tasking capable operating system for memory-efficient networked embedded systems and wireless sensor networks. It is easily portable to new platforms, and designed for micro controllers with small amounts of memory. A typical configuration using Contiki as operating system uses 2 KB RAM and about 40 KB of ROM. Contiki is already used in many different embedded systems, such as digital TV decoders, or wireless vibration sensors.

Supported micro controllers are the MSP430, the AVR, and old home computers. In detail, Contiki currently supports the following hardware platforms:

- *The Modular Sensor Board:* The Modular Sensor Board (MSB430) developed by FU-Berlin is supported by Contiki since spring of 2007. In its basic version, this board uses a Texas Instruments MSP430f1612 processor (introduced in Section 5.2.7), a Chipcon CC1020 radio chip, an MMA7260Q accelerometer, and a Sensirion SHT11 temperature and humidity sensor. The device also supports an SD device, thus the board is suitable for storage-centric sensor networks. In addition, there are other versions of the board that are equipped with a GPRS-modem and a GPS-receiver.

- *The ESB Embedded Sensor Board:* The ESB (Embedded Sensor Board) is a prototype wireless sensor network device developed at FU Berlin. It uses a Texas Instruments MSP430 low-power micro controller with 2k RAM and 60k flash ROM, a TR1001 radio transceiver, a 32k serial EEPROM, an RS232 port, a JTAG port, a beeper, and a number of sensors (passive IR, active IR sender/receiver, vibration/tilt, microphone, temperature).

- *The Tmote Sky Board:* The Tmote Sky platform is a wireless sensor board from Moteiv. It is also MSP430-based and equipped with an 802.15.4-compatible CC2420 radio chip, a 1 MB external serial flash memory, and two light sensors.

- *RZRAVEN LCD 3290p:* The Raven LCD Driver application software was designed for a user interface to the Contiki 6LoWPAN collaboration on board of the ATmega3290p.

- *RZRAVEN USB Stick (Jackdaw):* A USB stick with the overall idea to emulate an Ethernet interface. The Jackdaw can function as an 802.15.4 sniffer, and can sniff the raw 802.15.4 frame at the same time it is providing network functionality.

7.3.1 Networking

As Contiki introduced the idea of using IP communication in WSNs, an IETF standard and the IPSO Alliance, an internal industry alliance was build. Today, IP communication based on IPv4 and IPv6 (IPv6 Ready Phase 1 certified) is supported. Moreover, the kernel supports two different communication stacks:

- *uIP:* A TCP/IP stack that is RFC-compliant, to give the possibility to communicate also over the Internet. The stack contains the IP, ICMP, UDB and TCP protocols.
- *Rime:* A very lightweight communication stack that provides communication primitives, such as “best-effort local area broadcast”, or “reliable multi-hop bulk data flooding”.

To be able to use TCP/IP with Contiki, the uIP stack has been developed. Since with common sensor architectures only a few kilobytes of RAM is available, dynamic memory allocation is not used by the uIP implementation. Instead, a single global packet buffer to hold packets is used in combination with a fixed table for holding the connection state. This packet buffer is large enough to contain at least one packet of maximum size. This packet buffer will not be overwritten by new packets before the corresponding application has processed the data. It can also happen that packets arrive when the application did not finish the processing of old packets. Therefore, the new packet must be queued, either by the network device or by the device driver. Most network controllers have on-chip buffers, large enough to contain about 4 maximum sized Ethernet frames. If this is not sufficient, the packet is going to be dropped.

In uIP, there is only one packet buffer for incoming and outgoing traffic. This global buffer can also be used by application to temporary store data.

The amount of memory necessary for the uIP stack depends on the applications of the used embedded device. It is possible to run the uIP implementation with only 200 bytes of RAM, but such a configuration will provide low throughput and will only allow a small number of simultaneous connections.

To handle TCP connections very easily, the *protosocket* library provides an interface similar to the traditional BSD socket interface to the uIP stack. Unlike applications written for the uIP event-driven interface, applications written with the protosocket library are executed in a sequential order and do not have to be implemented as explicit state machines. This sequential control flow is provided by the usage of so-called *protothreads*, a type of lightweight stackless threads, designed for severely memory constrained systems such as WSNs. This makes the protosockets lightweight in terms of memory usage. However, protosockets inherit the functional limitations of protothreads, since each protosocket consists of a single function block. Automatic variables, such as stack variables, are not necessarily retained across a protosocket library function call.

In contrast to uIP, the Rime communication stack provides a set of lightweight communication primitives ranging from best-effort anonymous local area broadcast to reliable network flooding. The implemented protocols are build using a layered topology, with the more complex protocols implemented using less complex protocols. The type of communication in the Rime stack is based on scenarios to be used in typical sensor network protocols. Applications or even protocols running on top of the Rime stack can be attached at any layer of the stack and can use any of the communication primitives.

Both, single-hop and multi-hop communication primitives are supported by the Rime stack. To be able to implement arbitrary routing protocols on top of multi-hop primitives, the Rime stack does not specify how each packet is routed through the network. Instead, the application itself, or any upper layer protocol, is invoked on every node to choose the next-hop neighbor.

Moreover, every protocol or any application running on top of the Rime stack can implement other protocols that are currently not part of the Rime stack. If a protocol or application running on top of the Rime stack needs a communication primitive that is currently not in the stack, the application or protocol can implement the primitive on top of other communication primitives of the stack.

In addition to the communication stack, additional libraries can be linked to applications. These libraries could be a simple timer library or memory block management (described in Section 7.3.2). Furthermore, basic data structures, such as a linked list library, are available.

Access to the flash memory is possible using *Coffee*, a very simple, small, and easy to use file system. File access is handled based on C file access. Files can be read, written, or closed, Contiki manages the access to flash memory in background.

7.3.2 Process- and Resource management

Each process in Contiki consists of a single protothread, introduced in Section 7.3. The event driven Contiki kernel does not provide multi-threading, however, preemptive multi-threading is implemented as an additional library that optionally can be linked with user-applications. This additional library mainly consists of two parts:

- a platform independent part, which is the same for all platforms on which Contiki runs,
- and a platform specific part, which must be implemented specifically for the platform that the multi-threading library should run.

Without the additionally linked library, only rudimentary process-management is available, such as process_start, process_exit, process_post and so on. A Contiki subprocess is simply a process in a process.

In comparison to other embedded kernels, the Contiki kernel does not provide support for events based on timings. Nevertheless applications have the possibility to use timers by explicitly using the timer library. This library provides additional functions for setting, resetting, and restarting timers, and for checking if a timer has expired. It has to be mentioned, that each application must check on their own if its timers have expired. There is no automatism regarding timer-events. However a real-time module handles the scheduling and execution of real-time tasks (with predictable execution times).

Regarding the memory management of Contiki, memory blocks of fixed size have to be set. Therefore, allocation routines, such as memb_init, memb_alloc, and memb_free are provided. During compilation, a set of memory blocks is statically declared with the MEMB() macro. Memory blocks are allocated from the declared memory by the memb_alloc() function, and are deallocated with the memb_free() function. An additional available function of Contiki is its memory manager module that can be also linked to user applications. It keeps the allocated memory free from fragmentation by compacting the memory when blocks are freed. A program that uses the managed memory module cannot be sure that allocated memory stays in place. Therefore, a level of indirection is used: access to allocated memory must always be done using a special macro.

7.3.3 Protection

There is no security mechanism implemented in the Contiki operating system. Normally, the underlying hardware platforms does not support a MMU, thus, access to all memory-blocks are possible at any time.

7.3.4 Power Management

Contiki does not implement any power management functionality.

7.3.5 External Factors

Compared to other embedded operating systems, such as eCos (introduced in Section 7.2) the extension of Contiki in commercial or professional environments is much lower. Contiki is published under an Open Source, BSD-style license. The main programming language of Contiki is C.

Contiki is developed by groups of academic and industry partners, leaded by Adam Dunkels from the Swedish Institude of Computer Science. The development team also includes 16 additional developers from TU Munich, NewAE, Atmel, Cisco, SAP AG, and SICS.

At the moment, there is no certification of Contiki available and no commercial support offered. Two mailing lists are available:

- *contiki-commits* Automatically generated mails when CVS files are changed.
- *contiki-developers* For developers of Contiki applications and ports to new architectures.

Moreover, a project page http://sourceforge.net/apps/trac/contiki/ and a project Wiki http://sourceforge.net/apps/mediawiki/contiki/ is available. All necessary basic information can be found at the project homepage http://www.sics.se/~adam/contiki/.

At the moment, there are about 27 publications about Contiki published by different authors. Thus it can be stated that there is a relatively high circulation of the operating system at least in the scientific environment.

7.3.6 Total Cost Of Ownership

Since Contiki is licensed under the terms of a BSD-like license, there are no direct costs on purchasing the operating system. Indirect costs can hardly be estimated, since they heavily depend on the scenario of the project. As mentioned before, support of Contiki is only available via a mailing list.

7.3.7 Manageability

The documentation of Contiki is mainly based on doxygen[4], therefore detailed descriptions of each system function depends on the source-code documentation. Each data structure and function is described in a rudimentary but sufficient way. The user can find basic tutorials on the supported platforms, including instructions on how to start development. The Contiki Wiki is still in an early state but helps the user to find additional information about the operating system.

Contiki is using a BSD-like license, thus the source-code can be downloaded and modified regarding the user's scenarios. As mentioned in Section 7.3.8, a first entry to the system is very user friendly.

From the authors perspective, Contiki is sufficiently documented (but less, compared with other operating systems) to start development. The absence of a professional commercial support rises the danger of a long time continuation of this operating system, thus this operating system is used more in educational projects than commercial products. However, the usability of the system seems to be very good.

7.3.8 Using the operating system

The easiest way to build a development environment for Contiki is to download the pre-build image of a fully functional environment based on a Ubuntu Linux System. The Image is about 1 GB of size and can be download from http://www.sics.se/contiki/instant-contiki.html.

The image requires about 6 GB of disk space and a CPU of about 1,5 GHz. Moreover, about 2 GB of memory is required. The image can be executed using VMware Player (Windows only, http://www.vmware.com/products/player/) or VirtualBox (Windows, Linux, or Mac, http://www.virtualbox.org/wiki/Downloads).

After starting the virtual machine, the Log-In is easily possible with the user "user" and the password "user". Contiki is pre-installed and the development can start immediately.

More details on installing the Instant Contiki environment can be found at http://www.sics.se/contiki/instant-contiki.html

[4]See http://www.doxygen.org/index.html

7.4 TinyOS

Specially designed to be used in WSNs, TinyOS was developed during the last 5 years by the University of Berkeley. After several software iterations, TinyOS is now available for several sensor boards, such as:

- telos family
- micaZ,
- ITIS,
- mica2,
- the shimmer family,
- epic,
- mulle,
- tinynode,
- span, and
- iMote2.

Moreover, TinyOS supports the following micro controller platforms:

- the Texas Instruments MSP430 family,
- Atmel's Atmega128, Atmega128L, and Atmega1281, and
- the Intel px27ax processor.

Currently, support for the ARM Cortex M3 processor, which is similar to the Cortex M0 (see Section 5.2.1), is under development.

A typical TinyOS platform is equipped with 10 KB of RAM, 100 KB of ROM, and consumes 10 μA to 25 mA, depending on which components are currently active.

TinyOS features a small footprint with a low system overhead and low power consumption. It has a clear focus to support low-power operation, including wireless networking to be used in WSNs. These operations were also used in TinyOS network link layers (with single-hop or multi-hop communication). TinyOS also supports secure networking on specific network radio chips, such as the CC2420, of the leading 802.15.4/ZigBee radio chips. This support for network communication has become de-facto standard in the world of WSNs, also caused by the fact that many low-power wireless research groups that have released their code use TinyOS as their operating system.

In detail, TinyOS supports time synchronization in multi-hop scenarios using the FTSP protocol [MKSL04], data collection to a designated gateway or root with the CTP protocol [GFJ+09], reliable data delivery to all network nodes with the Trickle algorithm [LCH+], and installing new binaries over the WSN using the Deluge protocol [Hui].

7.4.1 Process- and Resource management

Based on an event-driven architecture, simple tasks can be implemented on top of sensor nodes using TinyOS. TinyOS uses the static programming language nesC in which all run-time memory usage is preallocated during compile-time.

Compared to other operating systems, TinyOS is less a classical kernel. The access to hardware is not granted by the kernel, but is managed directly by the user-applications. There is no process-management and only one process that runs on the fly. Also, there is no virtual memory allocation, thus TinyOS provides a single linear physical address space. The application memory has to be assigned at compile time, caused by the absence of a dynamic memory allocation.

There exists no exception handling or implementation of software signals, TinyOS is again focused to be as minimalistic as possible. Instead, the applications call functions to handle their proper fault-management.

Regarding the internal design, TinyOS is divided into different components. Every component provides and uses interfaces, which is the only way to interact with other components. These interfaces are bi-directional and declare a set of functions (called commands), the provider of the interface must provide. Function calls (events) used by the interface also have to be implemented. These components consist of:

- *Frame:* A Frame represents the internal state of a component.
- *Functions:* Functions are responsible for the computation of the application code itself.
- *Interface:* Interfaces include the events and commands of TinyOS components.

Moreover, TinyOS distinguishes between two different types of components:

- *Modules:* Modules represent the application code. They implement one or more interfaces to hard-/software of the system.
- *Configuration:* The component "configuration" is designed to connect other components together. It connects interfaces used by other components. This technique is called wiring.

Components and their interfaces are a set of commands and events. They define interaction boundaries between other components, commands as their entry points, and events as their callback points. The before mentioned component wiring provides functionality to all other components. Also there is no restrictions on usage of components – all components can use other components as they want.

Each nesC application is described by a top-level configuration, wiring together all its components inside. TinyOS does not block events at all, however, the latest version includes a thread library. In general, it is possible to run a threaded application on top of the rest of all other applications of the operating system. This threaded application now has a blocking API, and can also include large loops. This new library also allows the author to write applications in C, rather than nesC, what even reduces the effort or learning a new programming language.

7.4.2 Protection

In contrast to old versions of TinyOS, version 2.1 introduced some significant enhancements to core TinyOS components and interfaces. Special interest is the fully preemptable application-level threads library known as *TOSThreads*, and a runtime memory protection service called *Safe TinyOS* that enables the operating system to be able to make basic memory safety checks.

TOSThreads provides an extension to the pre 2.1 TinyOS concurrency model, requiring some changes to the TinyOS kernel In the existing TinyOS concurrency model, there were two execution contexts:

- synchronous (tasks), and
- asynchronous (interrupts).

Asynchronous code can preempt synchronous code but synchronous code is unable to preempt asynchronous code. TOSThreads provides a third execution context: *user-level application threads*. All threads are able to synchronize with each other and the system, using common synchronization techniques such as semaphores, mutexes, barriers, or condition variables. Switching the context produces an overhead of about 0.92 percentage, compared to applications without implemented threading techniques.

Current development also introduces *TinyLD*, a dynamic linker and loader for TinyOS. However the development is still not finished. After deployment, TinyLD can be used to dynamically deploy and execute TOSThreads based applications at runtime.

Optionally, TinyOS 2.1 applications can be used with its safe mode where the compiler is used to enforce memory safety at runtime. Using this mode, the programmer has to provide some extra annotations describing bounds of arrays and branches of unions. The compiler (Deputy compiler) then adds an additional check before each potentially unsafe operation. If the check fails, the application jumps to a fault handler. However, debugging with this feature is difficult, but in the past it helped finding unknown bugs as well as bugs that were already known. Safe TinyOS permits safe code to be freely mixed with unsafe code using new module-level nesC attributes *safe* and *unsafe*, with unsafe being the default.

Except these two protection mechanisms, TinyOS does not offer additional protection mechanisms.

7.4.3 Power Management

Since TinyOS is event-based, neither polling or blocking has to be handled, and therefore a more energy-efficient system can be designed. If the CPU is not used, cycles can be switched to sleep state, in contrast to actively waiting for an event.

7.4.4 External Factors

According to public available information, TinyOS is downloaded more than 35.000 times per year, what implicates that the operating system is widely used in different fields of application of industry and academic use. Official commercial users include Motorola, Intel, Arch Rock, Crossbow, and the People Power Company. From the authors perspective, it can be estimated that TinyOS is one of the most circulated operating systems in the area of WSNs. Since there is no commercial support available on the market, developers have to fall back to official mailing-lists.

The country of origin of TinyOS is the USA, University of Berkeley. As mentioned before, the current version is version 2.1.1, updates are published in regular intervals. TinyOS is written in nesC, a modularized programming language that provides structured component-based applications. NesC is an extension of the well-known C-language. Within TinyOS the new filename extension ".nc" is used. These "nesCurses" files are responsible for all source-files, interfaces, modules, and configuration.

7.4.5 Total Cost Of Ownership

There are currently no public details on a commercial usage of products based on TinyOS, thus an estimation of the total cost of ownership is not possible. In general, TinyOS is open-source and it can be used without any primary costs.

7.4.6 Manageability

There is a huge documentation and wide-spread community that helps interested users. The documentation is mainly published online (see [oB]).

TinyOS is open-source and uses the BSD-Licensing model. In contrast to the well known GPL-license, the TinyOS license does not require the developers to redistribute the source code. The license does have some restrictions, such as including copyright notices in documentation and not using the names of the developers to promote or endorse products. A copy with more details of this BSD-Licensing model is distributed with the source-code.

7.5 Reflex

REFLEX (Real-time Event FLow EXecutive) is an operating system implemented in C++ for embedded systems and wireless sensor nodes. The currently available version is 1.6. It is based on the event-flow principle, presented in [WN07]. This event-flow model was implemented in

C++ and allows to use common programming toolchains for development to lower the needed effort to port the system to new platforms. Currently supported controllers are:

- Atmel ATMega128,
- Freescale HCS12,
- Renesas H8/300,
- TI MSP430,
- normal PCs running Linux, and
- ARM based controllers.

Supported platforms including one of the above mentioned controllers are:

- CardS12,
- Mega128,
- Lega RCX,
- TmoteSky,
- OMNetPP,
- SERNet, and
- EZ430-Chronos.

The possibility of code-update is currently implemented in a basic version for the TI MSP430/TmoteSky as part of Reflex version 1.6. Future releases plan to integrate a user interaction interface.

7.5.1 Process- and Resource management

The Reflex operating system includes a scheduling framework which allows to choose between different schedulers, interrupt handling mechanisms, and event channels. Common I/O handling, memory management, power management, and virtual timer management are also included.

Since interrupt handling and synchronization of tasks need to be handled together, Reflex provides a common interrupt handling scheme. This code already implements the synchronisation with the system. As interface, the class *InterruptHandler* is given, which declares the abstract *handle()* method which has to be implemented by the specific handler.

Since in sensor systems have only a very little amount of RAM, Reflex does not support a classical memory management, thus the memory needs to be preallocated. However, Reflex supports different types of buffers, such as *SizedStack*, or *SizedFifoBuffer*. These buffers allow to pre-allocate memory space by instantiation of a pool and afterwards support dynamic access to it by allocating and freeing the corresponding buffers. These buffers are static allocated in size and number of elements at compile time. Moreover, the access to these buffers is optimized for communication usage (by including headers and trailers to each packet without copying any data). Since communication is asynchronous, it requires the presence of adequate buffers. In case of errors, references to a buffer can be kept in memory, e.g., for retransmission. When the reference count reaches zero, the buffer is implicitly freed again. In Reflex, a buffer can be used in two ways: as a stack and as a FIFO.

From version 1.6 on, Reflex supports so-called virtual timers. One hardware timer is virtualized to provide various components with timer events in millisecond precision. As an example, the most basic virtual timer available in all computer systems is the clock. Compared to previous versions of Reflex, the clock is now just another virtual timer, kept for legacy purposes.

With the introduction of virtual timers, Reflex does not use a static tick system but rather dynamic time ticks. The system will only wake on clock ticks when the next timer event is due or the hardware timer over-flows. Each internal component that needs a (virtual) timer input simply creates a new *VirtualTimer* object. Compared to the past, this virtual timer does not need anymore to be connected to any input sources or hardware timers. The timer will trigger the timer events automatically.

When an application needs even higher precision than the millisecond grained virtual timer can provide, another hardware timer on the corresponding platform must be implemented and used directly.

7.5.2 Power Management

In Reflex, so called activities are schedulable entities whenever a task was posted to its associated event buffer. Power Management in Reflex is divided into two views, the system view and the user view. The first is responsible to find the deepest sleep mode of the system. The second gives the programmer two different possibilities, groups and modes, to handle all hard and software components. Both views are shortly introduced in the following part.

- *System View:* Here, Reflex provides the class *EnergyManageAble*. Every instance of this class includes a variable, specifying the deepest possible sleep mode of the system. The implemented power manager includes a table of counters for every available sleep mode of the used microcontrollers. If a task is running, its possible sleep mode is signaled to the power manager by increasing its counter in the sleep mode table. If a task stops running, the counter is decreased again. If no task is running, the scheduler calls the *powerDown()* function.

- *User View:* During startup, each object is registered at the power management and assigned to one or more groups, defined by the programmer. This assignment allows a very easy way to enable or disable a several objects during runtime with only one method call. The groups also can be managed by predefined modes, dividing the execution of the application into different phases. The programmer is now responsible to switch between these modes, e.g., a times module could be implemented to change between modes.

In Reflex, there are two different types of interrupts. Those caused by external events, and interrupts as a result of software events. Therefore, the implemented power management is designed for interrupts caused by software events, since the drivers know when an interrupt has to be enabled. The decision which interrupt should be enabled is to be decided by the application programmer.

Regarding the Power consumption, [SWNN] shows a comparison of Reflex and TinyOS 2.0.2 on a TMoteSky [Corb] hardware. For the used application, Reflex consumed about 38 percent respectively 51 percent less energy, depending on the used system voltage.

7.5.3 External Factors

As programming language, C++ is used inside Reflex. The authors only found a few projects where Reflex was used [TCa, TCb], thus it can be estimated that Reflex is more or less only used in an educational environment. A commercial usage of this operating system could not be found. There is no certification on common assurance level criteria, the origin country of Reflex is Germany (Reflex is a project of the TU-Cottbus). The first official release was in the beginning of 2007, the latest version (version 1.6) was published in January 2010.

7.5.4 Total Cost Of Ownership

There is no market study available with real costs of the system, therefore no estimation could be given. However, the Lesser GNU General Public License is used, thus direct costs of licensing the product are not necessary.

7.5.5 Manageability

Most of the documentation is created by Doxygen. Also a wiki-based documentation on the installation procedure and further information on the supported platforms is available. Programmers in Reflex are requested to follow predefined code style guidelines, published on the official website of the project [TCf]. Also, two detailed PDF documents are available, describing details on programming Reflex, as well as system internals for further reading [TCc] and [TCg].

7.5.6 Using the operating system

As mentioned before, the current version of Reflex is 1.6. It can be downloaded either using the subversion repository [TCd] or using a tar.gz version [TCe]. Depending on the desired destination platform, the included Makefile needs to be adopted. Detailed information on each platform can be found at [TCf]. A template Makefile is already included in the package, therefore the following commands lead to a first compilation of the system.

Goto bin directory (applications$APPLICATION/platform$Platform/bin) and type

```
cp Makefile_default Makefile
vi Makefile

make clean
make all
make download
```

For compilation on an ARM platform, one needs to install the packages *binutils*, *gcc*, and *newlib*.

binutils:

```
wget http://www.gnuarm.com/binutils-2.16.1.tar.bz2
tar xjf binutils-2.16.1.tar.bz2
mkdir objdir
cd objdir
../configure --target=arm-elf --prefix=/usr/local/gnuarm --enable-interwork --enable-multil
make all
su
make install
```

gcc:

```
export PATH=/usr/local/gnuarm/bin:$PATH
wget http://www.gnuarm.com/gcc-4.0.2.tar.bz2
tar xjf gcc-4.0.2.tar.bz2
cd gcc-4.0.2
mkdir objdir
cd objdir
../configure --target=arm-elf --prefix=/usr/local/gnuarm --enable-interwork --enable-multil
make all-gcc
su
make install-gcc
```

newlib:

```
wget http://www.gnuarm.com/newlib-1.14.0.tar.gz
tar xzf newlib-1.14.0.tar.gz
cd newlib-1.14.0
mkdir objdir
cd objdir
../configure --target=arm-elf --prefix=/usr/local/gnuarm --enable-interwork --enable-multil
make all
su
make install
```

7.6 PikeOS

The PikeOS architecture, a real-time operating system developed by the German company SYSGO, is based on a small microkernel, providing a core set of services. Version 3.1 of PikeOS was released in March 2010, while version 3.2 was released in March 2011. Version 3.2 supports SMP on several platforms, along with performance improvements, and many enhancements in the offered system tools. The currently supported processors are:

- x86
- PowerPC
- MIPS
- ARM
- SPARC

PikeOS enables the so-called *Safe and Secure Virtualization (SSV)* technology that uses multiple operating system interfaces, called *Personalities*, to work on isolated separate parts of resources within a single device. These personalities can be native C/C++ programs, ARINC 653, Linux, POSIX, Certified POSIX, and RTEMS. Following SYSGO, a Windows Personality is currently being developed. However, PikeOS offers a wide range of features, but is still very simple and compact. The resulting design achieves a suitable real-time performance. Its support of multi-core architectures offers a flexible approach to the user who can select an execution model ranging from a pure AMP (Asymmetric Multi Processing) to full SMP.

An overview of the PikeOS architecture is shown in Figure 7.1

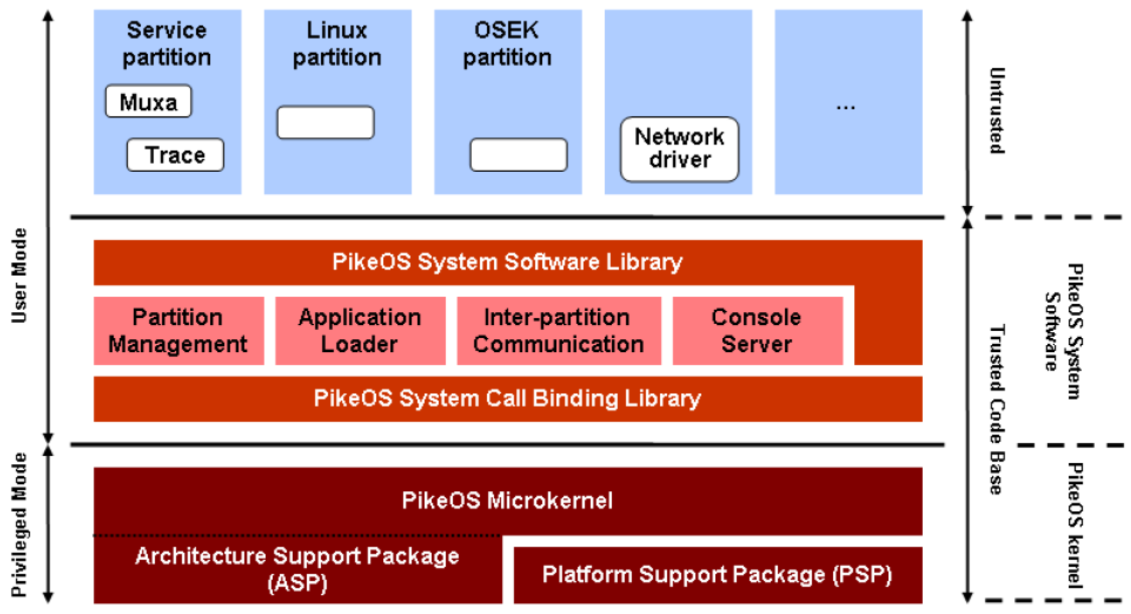


Figure 7.1: PikeOS architecture

7.6.1 Process- and resource management

Running applications inside an operating system that uses virtualization technologies requires a complex scheduling mechanism, since complex embedded systems must support a mixture of applications with a broad range of timing requirements: hard real-time, soft real-time, and non real time. Therefore, PikeOS offers a scheduler that is capable of combining time-driven and priority-driven scheduling. This scheduler ensures deterministic mapping between virtual-time and real-time, dynamic re-allocation of excess computing time and priority-based responsiveness.

Every virtual machine (here: Personality) is statically assigned an individual time slice. The virtual machine scheduler periodically executes each virtual machine in turn for the duration of their respective time slices. Therefore, each personality receives fixed amounts of processing capacity at predefined points in time. Thus, they are able to schedule real-time processes themselves. However, if a virtual machine has no runnable processes during its active time slice, or if its processes have completed before the time slice is over, it can not simply do a switch to another virtual machine.

In addition, PikeOS combines time-driven scheduling and priority-based scheduling. Each virtual machine can also be assigned with a specific priority. As an example, all real-time virtual machines receive the same mid-level priority. These virtual machines get time to process in a time-driven procedure. In contrast, all non-real-time virtual machines are assigned to a low priority. With this lower priority, the switching of each virtual machine is realized with a simple robin scheduler to achieve basic load balancing. In addition, if a real-time virtual machine (mid-level priority) has currently no processes to run, it sleeps and a switch to the next low-priority virtual machine is done for the remaining free process time of the mid-level virtual machine.

Within PikeOS, it is possible to create memory objects that are shared between multiple partitions. These shared memory objects have to be predefined at compile-time. They are created a boot time and were never deleted. A shared memory object does not belong to a specific partition: it can be accessed by any partition, as long as it was defined that it is allowed to access this memory object.

Using PikeOS, the existence of an MMU is essential. An MMU translates all virtual addresses of running code back into physical addresses. Moreover, for every memory access it is checked whether the corresponding task is allowed to access the specific part of memory, or not. If there is no permission to access, the MMU will generate an exception.

7.6.2 Protection

The partitioning concept of PikeOS is described in the ARINC 653 specification for system partitioning and scheduling. This specification is often required in safety-critical systems in the avionics industry. On top of the microkernel of PikeOS, multiple partitions are allowed to run in parallel. These partitions can contain real-time operating systems or run-time environments. Each partition receives its own set of system resources, fully isolated from each other. Each application operates completely isolated and is only controlled by the underlying microkernel. In theory, there is no way for a program in one partition to harm any application of another partitions. This idea of isolated processing is also currently used in desktop systems of security relevant areas.

The PikeOS microkernel only provides basic functionalities, such as memory management, access to attached devices, and granting CPU-time. It divides the underlying hardware to the above partitions. Therefore, it prevents to execute privileged application instructions through the implementation of para-virtualization.

A partition only has access to the memory and input-/output resources which are previously specified in the partition configurations. Memory is instantly available for the application upon startup. Memory that has been allocated to a partition will never be returned to the system, even if the partition itself is shut down. If the partition is later restarted, it will have access to exactly the same memory resources as before. This ensures:

- Predictive application behavior. Resources once available will not exhaust in a second run
- Information cannot be accidentally exchanged between partitions due to a re-assignment of physical memory pages.

PikeOS also provides a message-based communication through predefined communication channels. A communication channel is a link between two communication ports (source and destination). The source and destination can be located within the same partition, within two different partitions, or within one partition and a system extension.

7.6.3 External Factors

The basic design idea of PikeOS gives the possibility to be used in safety-critical applications and has validated according to safety standards, such as DO-178B, EN 50128, IEC 62304, IEC 61508, ISO 26262, IEC 61513 of either the avionics, automotive, railway, medical, industrial automation, or nuclear power plants. Since only the underlying microkernel is allowed to run in privileged mode, the kernel application directly contributes to the trusted code base of every application that runs on top of it. Therefore, the PikeOS microkernel consists of less than 10.000 lines of code making certification less expensive than that of conventional monolithic real-time operating systems. PikeOS currently is certifiable to safety standards such as DO-178B, IEC 61508 or EN 50128, it is Multiple Independent Level of Security (MILS) compliant, and is currently involved in various security standard CC EAL certification projects.

According to the MILS standard, PikeOS offers three levels of security. This MILS architecture offers a separation microkernel allowing the combination of trusted and untrusted code on a single hardware platform.

7.6.4 Total Cost Of Ownership

PikeOS is published using a proprietary license. A detailed pricing-model is not publicly available, therefore the total cost of ownership can not be estimated. However, since SYSGO offers commercial professional support of its operating system, it can be estimated that bugs will be fixed in an appropriate time. Also, missing ports to additional platforms could possibly be done by SYSGO.

7.6.5 Manageability

The package of PikeOS is shipped with a complete development environment called CODEO, an Eclipse-based IDE. CODEO makes graphical configuration tools available and offers a guided configuration, remote debugging (down to the hardware instruction level), target monitoring, remote application deployment, and timing analyses. In addition, CODEO provides standard application development features, such as compiler, assembler, and linker. The development environment includes

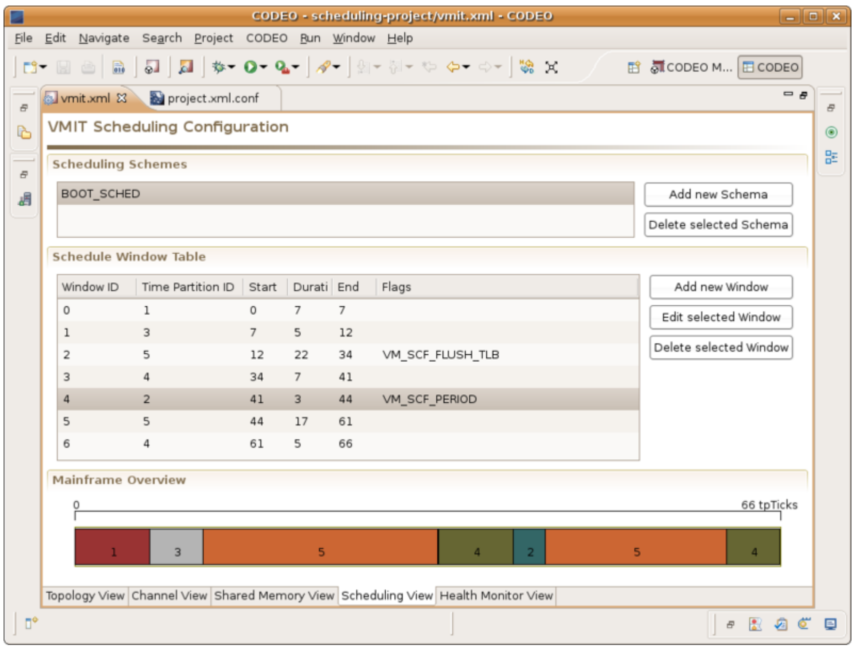
- project management,
- code browser,
- scheduling configurator
- configuration management, and
- interface components.

As an example of the CODEO feature, the scheduling configurator is shown in Figure 7.2.

Moreover, PikeOS with its CODEO offers the possibility to analyze the timing behavior of the developed applications. Trace points can be set, used as triggers, and extended using the graphical trace configuration and visualization editor, shown in Figure 7.3. Concurrent tracing is possible on multiple personalities.

Since debugging is sometimes essential, PikeOS also offers a debugging-tool called Muxa. With Muxa it is possible for external users to communicate with running applications through several communication channels. Also a trace tool is available to be able to trace schedule changes, system calls, interrupts, exceptions, and other relevant information.

Since PikeOS is not published under the terms of an open source license, the source-code is not publicly available, which prevents direct modifications of the source code. Therefore, a port to a new platform strongly depends on the manufacturer and the required efforts cannot be


CODEO - scheduling-project/vmit.xml - CODEO
File Edit Navigate Search Project CODEO Run Window Help
CODEO M... CODEO
vmit.xml
project.xml.conf

VMIT Scheduling Configuration

Scheduling Schemes

BOOT_SCHED

Schedule Window Table

Window ID	Time Partition ID	Start	Durati	End	Flags
0	1	0	7	7	
1	3	7	5	12	
2	5	12	22	34	VM_SCF_FLUSH_TLB
3	4	34	7	41	
4	2	41	3	44	VM_SCF_PERIOD
5	5	44	17	61	
6	4	61	5	66	

Figure 7.2: CODEO Scheduling Configurator

estimated. Additionally, since a proprietary license is used, no public documentation is available without buying PikeOS. Thus, there can be no conclusion about the quality of the documentation, shipped with PikeOS.

Public information of this analysis is mainly based on the Internet, press releases, and public available deliverables of projects, SYSGO is involved in. More details on PikeOS can also be found in `http://www.tecom-project.eu/downloads/deliverables2009/TECOM-D2.5-Comparison-\and-synthesis-PikeOS-and-Linux-personality.pdf`

7.6.6 Using the operating system

Although command-line tools are available, the common user is advised to use the before mentioned CODEO development environment. Command-line tools are available for Linux or through Cygwin. They are used for creating a new project, cloning an existing project, or building the project. If ELinOS, the Linux personality of PikeOS, is used, an additional tool called ELK (Embedded Linux Konfigurator) can be used to define which features should be included into a Linux partition. Possible features include network support, startup method parameters, or additional features. Also, ELK is used to modify the Linux kernel itself.

7.7 MANTIS

Around 2003, the MANTIS (MultimodAl system for NeTworks of In-situ wireless Sensors) was introduced to the public. It was designed to provide a multi-threaded embedded operating system that could be adopted to other sensor platforms, used for WSNs. However, since a lot of different embedded operating systems were already on the market, the designers of MANTIS tried to focus on the aggregation of processing complex tasks, such as compression, preemptive multi-threading, or signal processing together with time-sensitive tasks.

Since MANTIS is designed to be used within sensor nodes, it has to be very memory efficient. To achieve this memory efficiency, it is implemented in a way that it fits in less than 500 bytes of memory – including kernel, scheduler, and network stack. In addition, additional power-management features are offered by the operating system, described in Section 7.7.2. The two

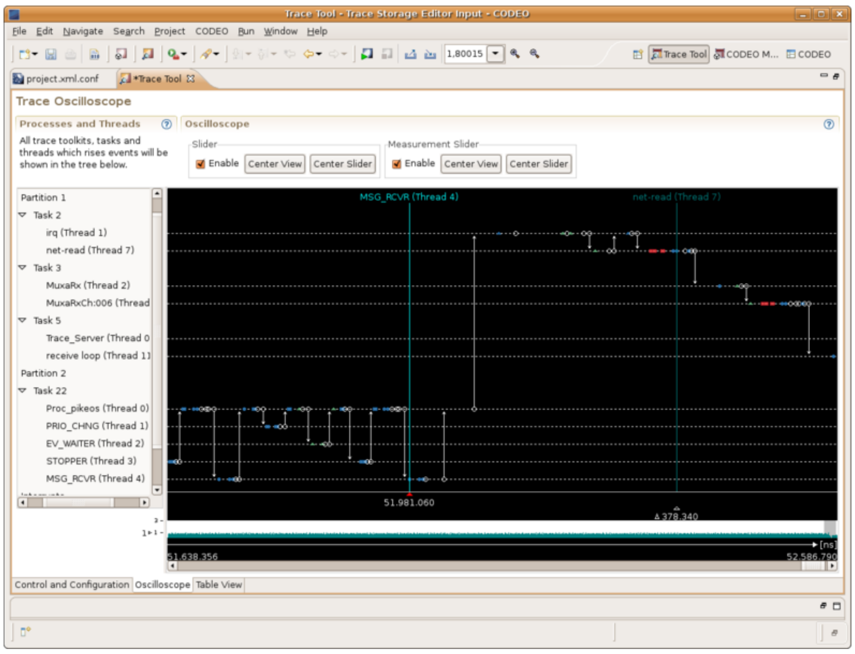


Figure 7.3: CODEO System Tracer

key-achievements are a lightweight memory footprint as well as an energy-efficient operation. Another interesting feature of MANTIS is its design to support remote management of in-situ[5] sensors via dynamic reprogramming and remote login.

MANTIS is by design very flexible regarding the supported hardware platforms, such that the same application code can be executed on different hardware platforms. MANTIS officially supports ATMEGA platforms, x86 platforms, and different micro sensor platforms, such as MICA2, Moteiv's Telos RevB, or MANTIS Nymph. The goal of MANTIS is also to support features, such as dynamic reprogramming of sensor nodes via wireless, remote debugging of sensor nodes, and multi-modal prototyping of virtual and deployed sensor nodes.

Figure 7.1 illustrates the general architecture of MANTIS, being a multi-threaded layered operating system, capable of running in less than 500 bytes of RAM.

Regarding the essential feature of code update within WSNs, the goal of MANTIS is to achieve dynamic reprogramming, flashing the entire operating system, reprogramming of a single thread, and changing variables within a thread. MANTIS also provides a remote shell that enables a user to login to a specific sensor to observe its memory of a running thread. Realization of support for dynamic reprogramming of the entire operating system is currently in progress.

7.7.1 Process- and resource management

MANTIS provides a subset of POSIX threads[6], a priority based thread-scheduling with round-robin semantics including priority levels. The operating system also supports binary (MUTEX) and counting semaphores. Each priority level has its own ready-list and tail pointer. There are five default priority levels consuming 20 bytes in total. The time-sliced multi-threading offers automatic preemption leading to a system where a single segment of application code cannot block the execution of other tasks. Multi-threading in MANTIS includes the necessity to switch context during run-time, which requires additional stack memory for each thread. However, the advantages of multi-threading preponderance the costs. Depending on the used

[5]In situ sensing, or local sensing, describes the process of measuring an object locally, with a sensing device that is closer to the object than the size of the sensor itself.

[6]See https://computing.llnl.gov/tutorials/pthreads/

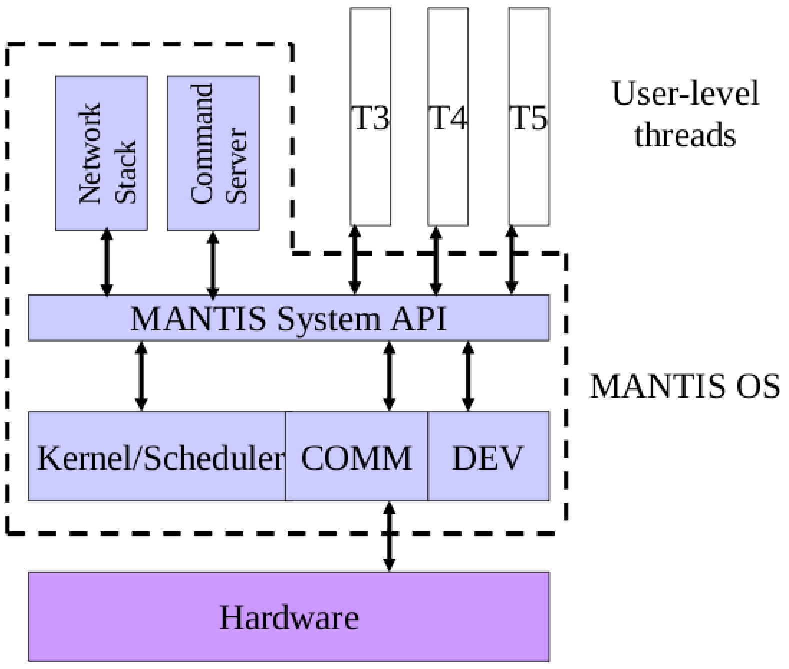


Figure 7.4: General MANTIS OS architecture

sensor architecture, each context switch requires about 60 microseconds (a representation of about 120 instructions), since 30 registers have to be stored and restored. The default stack size in MANTIS is 128 bytes.

MANTIS divides the RAM in two distinct sections: an area managed as a heap, and space for global variables, defined and allocated during compilation. Every new thread results in allocation of some space of the heap. If the thread finished, this memory is available for new threads. A dynamically allocation of the heap during run-time is not possible, caused by the fact that with very limited memory inside sensor nodes, memory management has to be very well planned.

The main data structure inside MANTIS is the global thread table, statically allocated during compilation, leading to the fact that the maximum number of threads has to be statically defined beforehand. Each table entry is 10 bytes and contains the stack pointer, base-pointer and size, the threads priority level, and the next thread pointer to be used as a linked list. The context of each thread needs only to be stored on its stack during it is allocated, thus the static overhead of the thread table is with 120 bytes very small. The MANTIS scheduler in total needs not more than 144 bytes.

The only interrupt, handled by the MANTIS kernel, is the timer interrupt. All other interrupts are directly forwarded to the associated device drivers.

One of the more user-level threads is the implemented network stack. It give the user the possibility to easily modify the network stack in user space and also decrease the amount of work needed to support cross-platform prototyping of network stack functionality. It supports layer three and above, thus network layer routing, transport layer, and application layer. Overall, the network stack of MANTIS consumes less than 200 bytes of RAM.

All implemented device drivers are located in the internal device layer. POSIX-style system calls are implemented for each device in a simple device layer. In addition, a single static table is used to store function pointer for each device implementation. This device scheme has been implemented for EEPROM, several assorted sensors, and is planned in the near future for accessing flash storage.

7.7.2 Power Management

As one of the few analyzed operating systems with a build-in power management, the following part describes the features of MANTIS to increase the energy efficiency:

To achieve energy efficiency, MANTIS modifies the internal scheduler. It sleeps the micro controller after all active threads have called the MANTIS function sleep(), resulting in a reduction of the current consumption to the μA range. To use the sleep() functions, all application threads have to enable power-save mode by calling the function mos_enable_power_mgt(). Also, each thread can be forced to sleep over a predefined period of time, realized with the method mos_thread_sleep(PERIOD). Combining both methods, MANTIS offers basic power management while maintaining a thread-capable scheduling mechanism. Figure 7.5 illustrates the general architecture of the MANTIS scheduler and its power management capabilities.

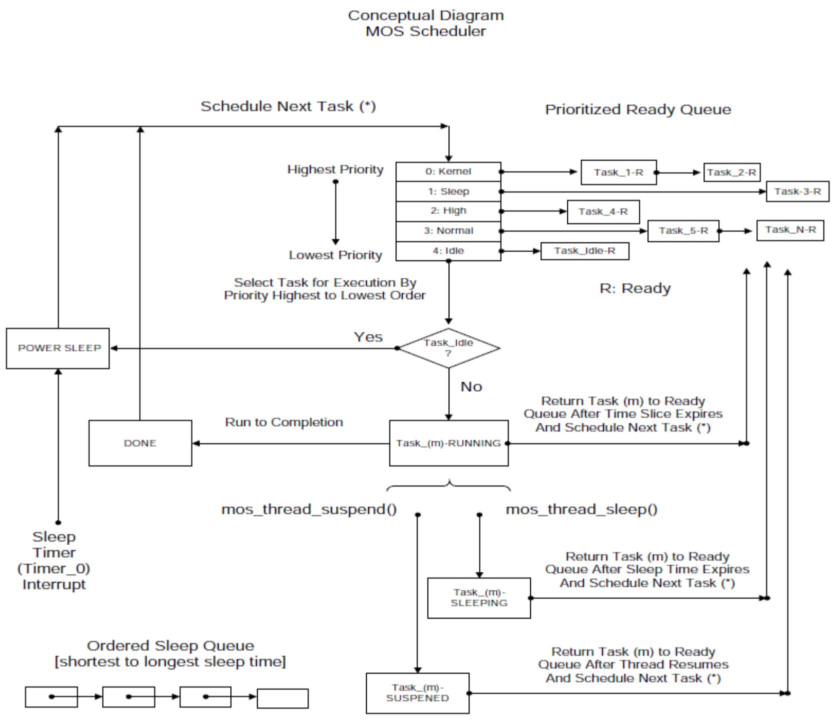

Conceptual Diagram
MOS Scheduler
Schedule Next Task (*)
Prioritized Ready Queue
Highest Priority
Lowest Priority
0: Kernel
1: Sleep
2: High
3: Normal
4: Idle
Task_1-R
Task_2-R
Task-3-R
Task_4-R
Task_5-R
Task_N-R
Task_Idle-R
Select Task for Execution By Priority Highest to Lowest Order
R: Ready
POWER SLEEP
Yes
Task_Idle ?
No
Return Task (m) to Ready Queue After Time Slice Expires And Schedule Next Task (*)
DONE
Run to Completion
Task_(m)-RUNNING
mos_thread_suspend()
mos_thread_sleep()
Sleep Timer (Timer_0) Interrupt
Return Task (m) to Ready Queue After Sleep Time Expires And Schedule Next Task (*)
Task_(m)-SLEEPING
Ordered Sleep Queue [shortest to longest sleep time]
Return Task (m) to Ready Queue After Thread Resumes And Schedule Next Task (*)
Task_(m)-SUSPENED

Figure 7.5: MANTIS scheduler architecture

In addition to threads created by drivers and users, MANTIS always runs an idle thread created by the kernel at startup. This idle thread uses the lowest priority and runs when all other threads are blocked. It gives the system the possibility to add power-aware mechanisms to increase the power efficiency of the sensor node running MANTIS by adjusting kernel parameters to save energy. However these adjusted parameters to lower the CPU performance are not included in the operating system. Regarding the ATMEL sensor node architecture, if all threads called sleep(), the system will go into the deep power-save sleep mode.

All implemented devices can run in three states: on, off, and idle. These states have to be enabled manually, also the resume of an idle device has to be handled by the application.

7.7.3 External Factors

The MANTIS kernel is written in the programming language C and developed by the wireless sensor networking research group at the University of Colorado, Boulder. The last changes are noted at January 2008, further changes of the system are not known, thus it can be expected that the circulation is more related to the educational area. This also leads to the fact that there is currently no certification according to some criteria.

7.7.4 Total Cost Of Ownership

Currently, there is no public study on costs of MANTIS, thus direct and indirect costs of MANTIS cannot be estimated in a proper way.

7.7.5 Manageability

The code of MANTIS is very compact. It allows moderate programmers with basic skills in C programming, rapidly create prototypes within a reasonable amount of time. As the authors of [BCD$^+$05b] mention, an application including a frequency-hopping protocol included with a port of the RC5 security standard would be implemented in less than tree days.

For interaction with the system and in-/output, MANTIS offers a set of system APIs. A complete documented list of the APIs can be found at [UoC]. Since MANTIS is optimized to support cross-platforms, the C programming language leads to the fact that no or only little adaptations need to be done when using a different platform.

In addition, MANTIS offers the possibility to be used inside a virtual network of sensor nodes. Therefore, applications can be evaluated in a virtual environment and later deployed to the real world. A combination is also possible, since virtual nodes can communicate with real nodes outside.

Current unstable versions of MANTIS are available under the BSD license. The current stable version (0.95) is available under an eCOS-style license.

Basic documentation of the API, troubleshooting, and installing how-tos are available. However, more detailed documentation is not provided.

7.7.6 Using the operating system

The complete source-code, including installation instructions, can be downloaded from the MANTIS home page[7]. As a first result, the authors admit that MANTIS currently is still work in progress. Due to this fact, it comes out that MANTIS is probably less of interest for usage within TeSOS, however we assume that in the near future the possible outcome of the operating system is very interesting. This is mainly resulted by the less frequent update cycle and the missing application in real world scenarios. Also missing public or commercial support argues against the usage of MANTIS within TeSOS.

7.8 Conclusion and suggested operating system to be used in TeSOS Integration

As an interactive process with ongoing discussions, BSI and TeSOS members came to the conclusion that for the Big Nodes (BNs), some more powerful operating system should be used. Therefore we suggest PikeOS (Section 7.6) or eCOS (Section 7.2). For the small nodes we suggest a minimalistic operating system such as TinyOS (Section 7.4). In the future, MANTIS (Section 7.7) is also interesting as a basis operating system for small nodes due to its included power management support. However, at the current status of development, the use of MANTIS cannot be suggested.

[7]http://www.mantisos.org

8 An Integrated Solution for Secure Sensor Networks (M6)

In this section, we discuss which of the components discussed in M3 to M5 can be combined sensibly to provide a comprehensive solution for TeSOS.

8.1 Sensor Base Platform

Operating System According to the analysis of Chapter 7, the two embedded operating systems MANTIS (see Section 7.7) and TinyOS (see Section 7.4) seem to be two adequate solutions to realize TeSOS: The main advantage of MANTIS is the support of power management, an important feature in the context of sensor nodes. However, the main disadvantages of MANTIS are the lack of commercial support and the long support cycles. In contrast, TinyOS has a big development community, short development cycles, and includes many additional software packages, e.g., for time synchronization, that are required for the intended security mechanisms. Therefore, we suggest at the moment to use the TinyOS operating system for both big nodes and small nodes.

If the Big Nodes (BNs) should support multitasking or execute trusted and untrusted code in parallel or isolate tasks for higher resilience against programming flaws, a microkernel-based operating system such as PikeOS or eCos should be chosen over MANTIS and TinyOS for better support of memory protection and isolation.

Hardware From the security standpoint, the Atmel SecureAVR (see Section 5.2.5) microcontroller is a good candidate for both big and small nodes, since it includes many security extensions such as a random number generator, hardware acceleration for common operations in RSA, DSA and ECC and secure storage. However, the SecureAVR is not supported by any of the analyzed operating systems and requires reimplementation or porting of the desired software stack.

Another good candidate is the Trusted Sensor Node (TSN, see Section 5.2.6). Its cryptographic co-processor supports a random number generator as well as ECC/RSA/SHA processing. However, the Leon processor of the TSN is currently only supported by the PikeOS and the eCos operating system.

If TinyOS is the operating system of choice due to its flexibility and functionality, only the TI MSP430 or Atmel's Atmega128x series can be used as the underlying hardware platform of TeSOS. However, these micro processors do not provide security extensions or cryptographic co-processors.

8.2 Security Mechanisms

For optimal interaction with Small Nodes (SNs), the BNs should support RSA acceleration for creating Rabin-Williams signatures, while ECC is more suitable for interaction between BN due to its shorter cipher text sizes. However, as discussed in previous section Section 8.1, only two hardware platforms are available that support ECC as well as RSA acceleration, but the

SecureAVR does not support commonly used embedded operating systems and the TSN is only a prototype and not suitable for production level systems.

In the following, we thus discuss two alternative approaches for the security concept of TeSOS. The first solution is optimized for a sensor network with implicitly trusted BNs. These BNs feature advanced security hardware for secure memory and ECC acceleration. In this scenario, we delegate critical security operations from the SNs to the BNs to reduce computation and communication loads, so that SNs are not required to process ECC ciphertext.

In the second scenario, we assume that the BN have the same or higher risk of compromise as the SN (i.e., they are not trusted), so that a delegation of security critical operations is not sensible. In this scenario, we thus aim to provide end-to-end security with all nodes at the cost of higher computation and communication load. For this purpose, the BNs are equipped with RSA acceleration that can be used to create signatures that the SNs can verify with reasonable costs.

8.2.1 Delegate Security to Trusted Big Nodes

If BN are trusted it is reasonable to delegate critical tasks to them to save energy consumption of SN and for the overall network. Hence, we delegate the authentication of broadcast messages, including Base Station (BS) commands and program code super-distribution, to the BN and use Message Authentication Codes (MACs) to protect the "last hop" from BNs to their respective Cluster Heads (BN). Moreover, super-distribution for SN can be cheaper since additional assumptions can be made on the availability of BNs. The detailed combination of the security protocols is outlined below.

Key Management For efficient and flexible key management in TeSOS, we suggest the combination of hardware-accelerated Elliptic Curve Cryptography (ECC) among BN as described in Section 6.2.3 combined with simple symmetric schemes for interaction between BN and SN. We exploit the assumed pre-deployment knowledge (see WSN.Deployment in Section 3.1.2) to deploy the LEAP master keys cluster-wise and establish keys between SNs of different clusters using key transport, as described in Section 6.2.3. The scheme yields a pair-wise key K between any two neighbor nodes in the network. In addition, a pair-wise shared key between BSs and each sensor node should be distributed during deployment. These pair-wise shared keys are never used directly but only to generate keys for specific protocols of other security mechanisms, such as symmetric authentication or encryption keys for secure communication.

Data Channels As discussed in Section 6.3, data channels can be divided into unicast channels from the sensor nodes (SN, BN) to BS and multicast channels from BS to the sensor nodes. As shown in Table 3.1 on page 17, exchanged information must be authenticated and fresh while encryption is optional. Hence, we suggest the use of end-to-end authentication for data sent to the BS as outlined in Section 6.3.1 but without complex data aggregation protocols since the BN can be trusted to aggregate data correctly. For data sent from the BS to the sensor nodes we suggest a hybrid broadcast authentication scheme as described in Section 6.2.5: Using ECC signatures for BNs and MACs for SNs, we can integrate hardware acceleration while at the same time reducing load on the SNs, which do not support hardware acceleration. We suggest to use time stamps as loosely synchronized freshness counters. They can also be used to provide time synchronization between nodes. Such synchronization is not very efficient due to network delay, but it can be enough for certain applications or be used to support the high-precision time synchronization discussed below. For encrypted communication between BS and other sensors, we distinguish unicast and multicast communication. For unicast communication, encryption keys can be derived from K. This is not practical for multicast communication, so we suggest to simply use link encryption in this case.

Super-distribution We suggest to use the Seluge protocol described in Section 6.3.3 for super-distribution of code images using ECC signatures for authentication. However, to mitigate the problem of signature verification on the SNs Seluge should only be executed between BNs and the BS. Once a BN received and validated a complete code image update (or other larger data chunks to be distributed in the network), it should inform each of its cluster nodes (SNs) about the update. Upon receiving this information, the contacted SN enters a special upgrade mode where it can receive the code image, authenticated by the pair-wise authentication key shared between SN and its Cluster Head (BN).

Software Integrity As discussed in Section 6.4, secure boot is rather expensive and hard to implement when considering hardware attacks. To validate the integrity of a remote sensor node, we thus suggest the use of Combined Hardware/Software Attestation as introduced in Section 6.5.1. The mechanism can be used when the integrity of a sensor is in question due to recorded misbehavior or when sensors are updated or re-programmed after undeployment. We emphasize that the proposed mechanism can also be implemented with simple keyed hash functions (MACs) instead of PUFs, if the tamper evidence property of PUFs is not required. In this case, the physical function can be easily simulated if the verifier knows the respective secret key, eliminating the problem of Challenge-Response Pair (CRP) generation and maintenance. On the other hand, pure software-based attestation can only be used if the verifier can assure the identity and hardware integrity of the prover and any undesired communication during attestation can be prevented reliably.

Secure Routing The TeSOS network should be clustered into *inner* and *exterior* clusters, as proposed in Section 6.6.1. Further, we suggest to use different routing protocol for inter- and intra-cluster communication.

Network clustering. We propose a double-cluster network representation, where network is clustered into *inner* and *exterior* clusters formed around respective cluster heads, as proposed in Section 6.6.1. For network clustering, location-based approach is feasible, where verification of BN locations is delegated to a main cluster head.

Intra-cluster routing. We propose the use of INtrusion-tolerant routing protocol for wireless SEnsor NetworkS (INSENS) [DHM03, DHM04, DHM06a] protocol for inter-cluster communication in TeSOS network. We suggest to modify the protocol as discussed in Section 6.6.3 to replace the weakly authenticated broadcast messages with unicast transactions.

Inter-cluster routing. We propose to use the Dynamic Window Secured Implicit Geographic Forwarding (DWSIGF) [HIJM09] protocol for inter-cluster routing. We require the following options for DWSIGF (cf. Section 6.6.2): (i) use the SIGF-2 protocol layer to maintain neighborhood-shared state and to enable cryptographic operations for authenticity, confidentiality, integrity, and freshness of messages, (ii) use random or multiple choice of the next hop candidate, (iii) omit transmission of location information in Request-to-Send (RTS) messages. Moreover, propose the following protocol modifications: (i) Broadcast authentication scheme to authenticate RTS messages as used in Secure Aggregation Protocol for Cluster-Based (SAPC) [BLM07, BLMB07], (ii) the location information which is used to make forwarding decisions should be certified by a trusted BS.

Time Synchronization The existing TinyReSync [SNW06] protocol can be used for time synchronization in TeSOS as presented in Section 6.7. The system requires only pair-wise shared authentication keys and can be used to establish the clock skew between two systems. Based on this, a common time can be established between individual nodes, clusters or in the complete WSN.

8.2.2 End-to-End Security with Untrusted Big Nodes

If the BNs have similar risk of becoming compromised as SNs, they should only be used as organizational backbone of the network infrastructure but not as explicit trust anchors. In this case, authentication of data in broadcast and super-distribution should be carried out independently by the respective connections endpoints and not be delegated to the BNs. In the following we provide a choice of alternative protocols that support this functionality at a reasonable energy consumption level. Note that the Software Integrity and Time Synchronization mechanisms do not specifically leverage the BNs and can be used as described in Section 8.2.1.

Key Management We propose to use the LEAP key management scheme to establish pair-wise shared keys between all sensor nodes, as outlined in Section 6.2.2. The protocol is highly suitable since TeSOS guarantees the secure deployment, initialization and upgrade of sensors as by the assumption ADV.OperationalPhase, Section 3.1.3. As a result, LEAP provides high resilience, scalability and extendability at low communication and computational costs. The pre-deployment knowledge of sensor locations(see WSN.Deployment, Section 3.1.2) can be used to deploy LEAP with different master keys to further increase its resilience to key compromise. Note that, if precise pre-deployment knowledge is given, an even simpler direct deployment of pair-wise keys can be used (see Section 4.2.1 and specifically [LN03c]) In this case, a few additional keys should be deployed to each sensor node for future extensions of the network. The scheme yields a pair-wise key K between any two neighbor nodes in the network. In addition, a pair-wise shared key between BSs and each sensor node should be distributed during deployment.

Data Channels and Super-Distribution For efficient broadcast authentication for SN despite the lack of hardware acceleration we suggest to leverage Rabin-Williams Signatures [Ber08]. As we discuss in Section 6.2.5, Rabin-Williams signatures are an optimized form of RSA signatures that allow very efficient signature verification at only slightly higher signature creation cost. The same approach can be adopted for the Seluge protocol to provide end-to-end authentication also for the super-distribution of data.

A general problem of the RSA signature approach is the rather large message size due to the large RSA modulus. However, the much more frequent upstream messages from sensors to the BS can still be secured using pair-wise authentication keys between BNs/SNs and the BS, same as commands intended only for small groups of sensors. The actual impact of broadcast authentication message size therefore depends on application and implementation and should be evaluated based on a prototype implementation or simulation.

Secure Routing We use a modified version of INSENS and DWSIGF with Rabin-Williams signatures to provide secure routing in face of untrusted Cluster Heads (CHs).

Network clustering. For network clustering, a location based approach can be used similar to the scheme described in Section 8.2.1. The location information provided by the BNs should be signed using Rabin-Williams signatures so that they can be verified by the SNs.

Intra-cluster routing. We suggest to use a modified INSENS protocol as discussed in Section 8.2.1 and Section 6.6.3. However, as we discuss in Section 6.6.3, the protocol is still vulnerable to DoS; no existing protocol can satisfy the TeSOS security requirements if BNs cannot assumed to be trusted. A novel protocol that fulfills these requirements is still work in progress.

Inter-cluster routing. The DWSIGF [HIJM09] protocol is suitable for inter-cluster routing in TeSOS with untrusted BNs, when modified as discussed in Section 8.2.1.

9 Conclusion

We reviewed and discussed the state-of-the-art hardware, software and security technology available for sensor networks. With regards to the specific requirements of the TeSOS scenario considered in this work, it becomes apparent that certain areas such as software integrity and secure routing are largely unexplored while other areas such as key management and (mostly) insecure routing protocols were discussed in great detail. Similarly, we note that few hardware platforms support security extensions such as random number generators or hardware acceleration of modern cryptographic algorithms, such as AES or SHA-1, and only the SecureAVR provides a type of secure memory.

We identify the problems of hardware compromise, remote software integrity verification and secure routing as the major problems in secure sensor network design. We provided first steps toward software integrity verification and secure routing but cannot address the problem of hardware compromise. Based on our analysis we propose to extend secure sensor hardware with secure memory that is directly attached to symmetric and asymmetric cryptographic accelerators to reduce the risk of key compromise during hardware attacks. PUFs present an interesting option for the implementation of such mechanisms.

Glossary and Acronyms

Advanced Encryption Standard (AES) The Advanced Encryption Standard (AES) is an encryption standard recommended by the National Institute of Standards and Technology (NIST). 43, 53, 55, 57, 58, 60, 62–64, 67, 69, 71–73, 77

Advanced Microcontroller Bus Architecture (AMBA) The Advanced Microcontroller Bus Architecture was introduced in 1996 and is widely used as the on-chip bus for ARM processors. [ARMa]. 49, 50, 52, 58

Advanced Peripheral Bus (APB) APB is designed for low bandwidth control accesses, for example register interfaces on system peripherals. This bus has an address and data phase similar to AHB, but a much reduced low complexity signal list, for example no bursts [ARMa]. 49, 50

AMBA High-performance Bus (AHB) AHB is a bus protocol introduced in Advanced Microcontroller Bus Architecture version 2 published by ARM Ltd company [ARMa]. 48–50, 58

Analog-to-Digital Converter (ADC) A device which converts continuous analog signals to discrete digital numbers. 59, 113

Application Programming Interface (API) A particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. 112–114, 116, 136

Application-Specific Integrated Circuit (ASIC) An application-specific integrated circuit (ASIC) is an Integrated Circuit (IC) customized for a particular use, rather than intended for general-purpose use. 49, 53

Arithmetic Logic Unit (ALU) In computing, an arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. 57

ARM Holdings (ARM) ARM Holdings (LSE: ARM, NASDAQ: ARMH) is a technology company headquartered in Cambridge, England, UK. 45–48

Asymmetric Multiprocessing (AMP) A concept for multiple-CPU computing where some CPUs have different capabilities and/or purpose. 51, 52

Base Station (BS) Centralized control and aggregating data center. Typically Base Station serves as a gateway to another network, a powerful data processing or storage center, or an access point for a human interface. 7, 13–15, 25, 37, 88, 89, 91–93, 95, 96, 100, 101, 103, 104, 138–140

Berkeley MAC (B-MAC) . 21, 22

Big Node (BN) Sensor network node with comparably high computational performance and energy reserves. 13, 41, 46, 53, 55, 86, 87, 89, 90, 93, 95, 96, 100–105, 136–140

Bundesamt für Sicherheit in der Informationstechnik (BSI) The Bundesamt für Sicherheit in der Informationstechnik (abbreviated BSI - in English: Federal Office for Information Security) is the German government agency in charge of managing computer and communication security for the German government. 45

Central Processing Unit (CPU) The Central Processing Unit (CPU) or the processor is the portion of a computer system that carries out the instructions of a computer program, and is the primary element carrying out the computer's functions. 40, 44, 47, 51–55, 68, 70, 106–110, 115–117, 125, 130, 135

Certificate Authority (CA) A trusted entity that certifies public keys of communication parties to assign them a valid identity in the network. 31

Challenge-Response Pair (CRP) A pair of values that correspond to each other w.r.t. a specific task. 99, 139

Clear-to-Send (CTS) Clear-to-Send message is a part of the two-way request-to-send/clear to send (RTS/CTS) handshake that many MAC protocols use to mitigate the hidden node problem. 102, 103

Cluster Head (CH) Nodes in WSN carrying out special functions such as aggregation of data within clusters and inter-cluster communication. 7–9, 13, 28, 32, 34, 37, 86, 92, 93, 100, 101, 104, 140

Common Criteria (CC) The Common Criteria for Information Technology Security Evaluation (abbreviated as Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security certification. 44

Complementary Metal-Oxide Semiconductor (CMOS) A technology for constructing integrated circuits using complementary transistor arrangements for low static power consumption. 53

Complex Instruction Set Computer (CISC) An instruction set architecture in which each instruction can execute several low-level operations in a single instruction. 53

Controller Area Network (CAN) A vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer. 113

Coordinated Universal Time (UTC) A universal time standard that tracks mean solar time on Earth. 36

Cyclic Redundancy Check (CRC) A hash function designed to detect accidental changes to raw computer data, and is commonly used in digital networks and storage devices such as hard disk drives. 113

Debug Support Unit (DSU) The Debug Support Unit (DSU) is an in-circuit emulator (ICE), a hardware device used to debug the software of an embedded system. It was historically in the form of bond-out processor which has many internal signals brought out for the purpose of debugging. These signals provided information about the state of the processor. 50

Denial of Service (DoS) A class of attacks on a network that is designed to make network services or resources unavailable. 16, 18, 21, 23, 27, 32, 94, 104

Differential Electromagnetic Analysis (DEMA) Differential electromagnetic analysis is a more advanced form of electromagnetic analysis which can allow an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations. 56

Differential Power Analysis (DPA) Differential power analysis (DPA) is a more advanced form of power analysis which can allow an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations. 56

Digital Encryption Standard (DES) The Digital Encryption Standard (DES) is an (outdated) encryption standard. 53, 55, 56, 60

Digital Rights Management (DRM) A generic term for access control technologies that can be used by hardware manufacturers, publishers, copyright holders and individuals to impose limitations on the usage of digital content and devices. 43

Digital Signal Processor (DSP) A processor that is optimized for digital signal processing. 32, 40, 44, 62

Digital Signature Algorithm (DSA) The Digital Signature Algorithm (DSA) is a signature standard recommended by the National Institute of Standards and Technology (NIST). 55

Direct Memory Access (DMA) A technology that allows devices to get direct access to computer memory, reducing CPU usage and thus increasing data throughput. 53, 54

Direct Sequence Spread Spectrum (DSSS) Method of transmitting radio signals. The data being transmitted is multiplied by a "noise" signal. This noise signal is a pseudorandom sequence of 1 and -1 values, at a frequency much higher than that of the original signal, thereby spreading the energy of the original signal into a much wider band. This noise-like signal can be used to exactly reconstruct the original data at the receiving end, by multiplying it by the same pseudorandom sequence. 21

Dynamic Window Secured Implicit Geographic Forwarding (DWSIGF) . 33, 102, 103, 139, 140

Electrically Erasable Programmable Read-Only Memory (EEPROM) A type of non-volatile memory used in computers and other electronic devices to store small amounts of data that must be saved when power is removed, e.g., calibration tables or device configuration. 19, 42, 53, 55, 56, 61

Electronic Code Book (ECB) A method of using block ciphers to encrypt data that is longer than the block length of the cipher. 58

Elliptic Curve Cryptography (ECC) A class of public key cryptography algorithms based on point-multiplication. 13, 32, 43, 55, 57, 58, 72, 87, 91, 138, 139

Ephemeral Diffie-Hellman (EDH) A Diffie-Hellman (DH) key exchange using ephemeral DH keys for both involved parties. 86

Federal Information Processing Standards (FIPS) Federal Information Processing Standards (FIPS) are publicly announced standards developed by the United States federal government for use by all non-military government agencies and by government contractors. 44

Field-Programmable Gate Array (FPGA) A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing. 49, 53

File Transfer Protocol (FTP) A standard network protocol used to copy a file from one host to another over a TCP-based network, such as the Internet. 113

floating-Point Unit (FPU) A floating-point unit is a part of a computer system specially designed to carry out operations on floating point numbers. 49, 52

Frequency-Hopping Spread Spectrum (FHSS) Method of transmitting radio signals by rapidly switching a carrier among many frequency channels, using a pseudorandom sequence known to both transmitter and receiver. 21

Full Function Device (FFD) Wireless sensor node with ability to forward messages via network. 8

General Purpose Input/Output (GPIO) An interface available on some devices. A microprocessor, microcontroller or interface device may have one or more GPIO connections to interface with external devices and peripherals. 41

Global System for Mobile communications (GSM) GSM (Global System for Mobile Communications: originally from Groupe SpÃ©cial Mobile) is the most popular standard for mobile telephony systems in the world. 55

GNU General Public Licence (GPL) The GNU General Public License (GNU GPL or simply GPL) is a widely used free software license. 51

GNU Lesser General Public Licence (LGPL) The GNU Lesser General Public License (formerly the GNU Library General Public License) or LGPL is a free software license published by the Free Software Foundation (FSF). 48

Greedy Perimeter Stateless Routing (GPSR) Geographic routing algorithm for wireless sensor networks. 33

Hardware Abstraction Layer (HAL) An abstraction layer, implemented in software, between the physical hardware of a computer and the software that runs on that computer. 112–115, 117, 118

Harvard architecture (Harvard architecture) Features physically separate storage for instructions and data. 20

Heterogeneous Wireless Sensor Network (HSN) Wireless sensor networks with heterogeneous hardware. 33, 34, 100

Identity-based Encryption (IBE) An asymmetric encryption scheme where an arbitrary text string can be used as public key. 32

Implicit Geographic Forwarding (IGF) . 33

Instruction Set Architectures (ISA) An instruction set is a list of all the instructions, and all their variations, that a processor (or in the case of a virtual machine, an interpreter) can execute. 44

Integrated Circuit (IC) In electronics, an integrated circuit (also known as IC, microcircuit, microchip, silicon chip, or chip) is a miniaturized electronic circuit (consisting mainly of semiconductor devices, as well as passive components) that has been manufactured in the surface of a thin substrate of semiconductor material. 142

Inter-Integrated Circuit (I^2C) Also called I^2C, a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone. 41, 53, 59, 112, 148

Intrusion Detection System (IDS) An agent designed to detect malicious presence and, if possible, react appropriately to prevent the intruder from accessing or damaging the system. 25, 28, 38, 39

INtrusion-tolerant routing protocol for wireless SEnsor NetworkS (INSENS) . 34, 103, 104, 139, 140

Joint Test Action Group (JTAG) A standard for testing and debugging of electronic devices, published as IEEE standard 1149.1. 53

Key Distribution Center (KDC) A trusted party that securely generates and distributes keys to other systems. 87, 89

Lightweight MAC (LMAC) An energy-efficient lightweight medium access protocol for wireless sensor networks. 21

Localized Encryption and Authentication Protocol (LEAP) A key distribution protocol for WSN that supports link- and cluster-keys and allows for later extensions of the network with additional nodes. 87, 88

Mean Time Between Failures (MTMF) The arithmetic mean (average) time between failures of a system. 41

Media Access Control (MAC) The Media Access Control (MAC) data communication protocol sub-layer, also known as the Medium Access Control, is a sublayer of the Data Link Layer specified in the seven-layer OSI model (layer 2). 21, 22, 37, 49, 50

Memory Management Unit (MMU) A memory management unit, sometimes called paged memory management unit (PMMU), is a computer hardware component responsible for handling accesses to memory requested by the central processing unit (CPU). 50, 52, 109, 121, 130

Message Authentication Code (MAC) A cryptographic primitive that provides message authentication. Typically implemented as a Hash-MAC, where the message concatenated with a secret authentication key are fed to a cryptographically secure hash function to generate an authentication token for the respective message called MAC or tag.. 34, 92–94, 104, 138

Million Instructions Per Second (MIPS) A well-known measure for the speed for a CPU. 51, 53, 55, 57, 59

MultiMediaCard (MMC) A flash memory memory card standard. 113

Nested Vectored Interrupt Controller (NVIC) An interrupt controller of ARM embedded Cortex processors. 47, 48

One-Time Programmable (OTP) One-time programmable, a type of programmable read-only memory (PROM) in electronics. 56

Peripheral Component Interconnect (PCI) Conventional PCI (part of the PCI Local Bus standard and often shortened to PCI) is a computer bus for attaching hardware devices in a computer. 49, 50

Personal Digital Assistant (PDA) Mobile device typically used to organize a person's life by taking notes, holding contacts, and connecting to the Internet. It is also known as a palmtop computer. 13

Physically Unclonable Function (PUF) A function that can not be cloned physically. 13, 28, 90, 96, 98

Programmable Read-Only Memory (PROM) A form of digital memory where the setting of each bit is locked by a fuse or antifuse. Such PROMs are used to store programs permanently. The key difference from a strict ROM is that the programming is applied after the device is constructed. 50, 61

Pseudo-Random Function (PRF) An idealized cryptographic primitive that deterministically maps any input to an output string of fixed length, such that it is arbitrary hard to find collisions, i.e., inputs that result in the same output. 30

Pseudo-Random Number Generator (PRNG) A pseudorandom number generator (PRNG) is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. 13, 57, 58

Public Key Cryptography (PKC) Cryptographic methods to transform messages using asymmetric key algorithms. 102, 103

Random Access Memory (RAM) Random-access memory is a form of computer data storage. 50, 55, 57, 58, 61, 62, 68, 112, 126, 133, 134

Random Number Generator (RNG) A computational or physical device designed to generate a squence of numbers or symbols that lack any pattern, i.e. appear random. 55, 56

Read-Only Memory (ROM) Read-only memory is a class of storage media used in computers and other electronic devices. 56, 61

Real-Time Clock (RTC) A real-time clock (RTC) is a computer clock (most often in the form of an integrated circuit) that keeps track of the current time. 53–55

Reduced Function Device (RFD) Wireless sensor node without ability to forward messages via network, such nodes typically take measurements. 8, 9

Reduced Instruction Set Computer (RISC) A CPU design strategy based on the insight that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction. 48, 53, 56, 58

Request-to-Send (RTS) Request-to-Send message is a part of the two-way request-to-send/clear to send (RTS/CTS) handshake that many MAC protocols use to mitigate the hidden node problem. 102, 103, 139

Resilient Geographic Routing (RGR) . 33, 102, 103

Resource Oriented Security Solution (ROSS) A framework to protect a network layer of heterogeneous sensor networks from attacks. 34, 100

Scalable Processor Architecture (SPARC) SPARC (from Scalable Processor Architecture) is a RISC instruction set architecture (ISA) developed by Sun Microsystems and introduced in 1986. 49–53, 67

Secure Aggregation Protocol for Cluster-Based (SAPC) A Secure Aggregation Protocol for Cluster-Based Wireless Sensor Networks. 93, 103, 139

Secure Configuration Register (SCR) A register of a CPU with ARM TrustZone. This register and its "NS bit" directly represent if the SCR corresponds to the secure or non-secure virtual processor. 46

Secure Digital Input Output (SDIO) Secure Digital (SD) is a non-volatile memory card format developed by Matsushita, SanDisk, and Toshiba for use in portable devices. An SDIO card is a combination of an SD card and an I/O device. 41

Secure Groupwise Synchronization (SGS) A group-wise time synchronization protocol. 37

Secure Hash Algorithm-1 (SHA1) In cryptography, SHA-1 is a cryptographic hash function designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard. 43, 57, 58, 62–64, 67, 69, 71, 72, 82

Secure Implicit Geographic Forwarding (SIGF) . 33

Secure Pairwise Synchronisation (SPS) A pair-wise time synchronization protocol. 36, 37, 104, 105

Secure SPIN (S-SPIN) Secure counterpart of the SPIN protocol. 34

Secure Transitive Multi-hop Synchronization (STM) A multi-hop time synchronization protocol. 37, 104

Sensor MAC (S-MAC) . 21, 22

Sensor Protocols for Information via Negotiation (SPIN) A family of adaptive protocols for sensor networks with flat topology. 34

Serial Peripheral Interface Bus (SPI) A synchronous serial data link standard named by Motorola that operates in full duplex mode. 41, 59, 112

Simple Electromagnetic Analysis (SEMA) Simple electromagnetic analysis involves visually interpreting electromagnetic traces, or graphs of electrical activity over time. 56

Simple Network Time Protocol (SNTP) A protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks. 113

Simple Power Analysis (SPA) Simple power analysis (SPA) involves visually interpreting power traces, or graphs of electrical activity over time. 56

Single Event Upset (SEU) A single event upset (SEU) is a change of state caused by ions or electro-magnetic radiation striking a sensitive node in a micro-electronic device, such as in a microprocessor, semiconductor memory, or power transistors. 49

Single instruction multiple data (SIMD) Single instruction, multiple data (SIMD), is a class of parallel computers in Flynn's taxonomy. It describes computers with multiple processing elements that perform the same operation on multiple data simultaneously. 45

Small Node (SN) Sensor network node with very little computational performance and energy reserves. 13, 41, 46, 47, 53, 60, 86, 87, 89–93, 95, 96, 100–102, 104, 105, 137–140

Static Random Access Memory (SRAM) Static RAM, RAM that does not need periodic refresh like Dynamic RAM. 19, 41, 50, 53, 54, 61, 69

Subscriber Identity Module (SIM) A subscriber identity module (SIM) on a removable SIM card securely stores the service-subscriber key (IMSI) used to identify a subscriber on mobile telephony devices (such as mobile phones and computers). 55

successive approximation ADC (SARADC) A type of analog-to-digital converter that converts a continuous analog waveform into a discrete digital representation via a binary search through all possible quantization levels before finally converging upon a digital output for each conversion. 59

Synchronous Dynamic Random Access Memory (SDRAM) Synchronous dynamic random access memory (SDRAM) is dynamic random access memory (DRAM) that has a synchronous interface. 41, 50, 61

Synchronous Multiprocessing (SMP) A concept for multiple-CPU computing where all CPUs run identical hardware. 51, 52, 112, 114, 116, 129

System Management Bus (SMB) The System Management Bus (abbreviated to SMBus or SMB) is a single-ended simple two-wire bus for the purpose of lightweight communication. 53

Timeout MAC (T-MAC) . 22

Tiny Lightweight Underlay Ad hoc Routing protocol (tinyLUNAR) A routing protocol for wireless sensor networks which supports multiple communication paradigms at the same time: Data-centric, geographic-based and address-centric. 34

Total cost of ownership (TCO) A financial estimate whose purpose is to help consumers and enterprise managers determine direct and indirect costs of a product or system. 111

Translation Look-aside Buffer (TLB) A translation lookaside buffer (TLB) is a CPU cache that memory management hardware uses to improve virtual address translation speed. It was the first cache introduced in processors. 50

Trusted Computing Group (TCG) An International standards group focusing on the specification and promotion of Trusted Computing standards. 35

Trusted Platform Module (TPM) A hardware device, protected against manipulation and designated for opt-in usage, providing protected capabilities and shielded locations. The TPM is a passive component and contains engines for random number generation, calculation of hash values and RSA key generation. A TPM generates and stores keys, signs or binds data to the platform and measures the platform's current state. 35, 42, 43, 96

Trusted Sensor Node (TSN) A sensor node based on the Leon2-Architecture, enhanced with additional cryptographic hardware like AES, ECC and SHA1. 45

Two Wire Interface (TWI) TWI-Bus or Two Wire Interface, a variant of Inter-Integrated Circuit (I^{2}C). 53

Two-Tier Secure Routing (TTSR) A secure protocol for heterogeneous wireless sensor networks. 34, 100

Universal Asynchronous Receiver/Transmitter (UART) A type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. 41, 49, 50, 53, 59

Universal Serial Bus (USB) A serial bus standard to connect devices to a host computer. . 41, 113

Very-high-speed integrated circuits Hardware Description Language (VHDL) VHDL is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. 48, 49, 72

Virtual Local Area Network (VLAN) A group of hosts with a common set of requirements that communicate as if they were attached to the same broadcast domain, regardless of their physical location. 42

Von Neumann architecture (VNA) Features a single physical storage structure for storing both code instructions and data. 20

Wake-up Interrupt Controller (WIC) An interrupt controller of ARM embedded Cortex processors, providing interrupt detection logic while the processor is in sleep mode. 47, 48

Wireless Local Area Network (WLAN) A wireless local area network (WLAN) links devices via a wireless distribution method (typically spread-spectrum or OFDM) and usually provides a connection through an access point to the wider Internet. 41

Wireless Sensor Network (WSN) Wireless network build up from a set of sensor nodes. Typically with random physical topology. 5–8, 10, 12–16, 18–25, 27–39, 41–44, 47, 48, 50, 51, 53, 55, 57, 58, 60–63, 86, 89, 92–94, 96, 97, 100, 103, 105, 106, 110, 111, 120, 123, 125, 132, 133

ZigBee (ZigBee) A specification for a suite of high level communication protocols based on the IEEE 802.15.4 standard for wireless personal area networks. 26

Bibliography

[Á09] Gergely Ács. Secure-TinyLUNAR: A secure label-switching routing protocol for wireless sensor networks. In *FuturICT'09: Hungarian-Japanese Joint Conference on Future Information and Communication Technologies*, 2009.

[ABV06] Gergely Ács, Levente Buttyán, and István Vajda. Modelling adversaries and security objectives for routing protocols in wireless sensor networks. ACM, 2006.

[ABV07] Gergely Ács, Levente Buttyán, and István Vajda. The security proof of a link-state routing protocol for wireless sensor networks. In *Workshop on Wireless and Sensor Networks Security (WSNS'07)*. IEEE, 2007.

[Aer] Aeroflex Gaisler. GRLIB IP Library. `www.gaisler.com/cms/index.php?option=com_content&task=section&id=13&Itemid=125`.

[AFS97] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A secure and reliable bootstrap architecture. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 1997.

[AGD08] Piyush Agrawal, R. K. Ghosh, and Sajal K. Das. Cooperative black and gray hole attacks in mobile ad hoc networks. In *International conference on Ubiquitous information management and communication (ICUIMC)*. ACM, 2008.

[AGKL05] Nael Abu-Ghazaleh, Kyoung-Don Kang, and Ke Liu. Towards resilient geographic routing in WSNs. In *Q2SWinet '05: International workshop on Quality of service & security in wireless and mobile networks*. ACM, 2005.

[AHK04] Imad Aad, Jean-Pierre Hubaux, and Edward W. Knightly. Denial of service resilience in ad hoc networks. In *International Conference on Mobile Computing and Networking, (MobiCom)*. ACM, 2004.

[AKFS98] William A. Arbaugh, Angelos D. Keromytis, David J. Farber, and Jonathan M. Smith. Automated recovery in a secure bootstrap process. In *Network and Distributed Systems Security (NDSS)*. Internet Society, 1998.

[Ala] Alan R. Weiss. Dhrystone Benchmark - History, Analysis, "Scores" and Recommendations. `www.johnloomis.org/NiosII/dhrystone/ECLDhrystoneWhitePaper.pdf`.

[AMP08] F. Amini, J. Misic, and H. Pourreza. Detection of sybil attack in beacon enabled ieee802.15.4 networks. In *International Wireless Communications and Mobile Computing Conference (IWCMC'08)*. IEEE, 2008.

[AQ05] O. Arazi and Hairong Qi. Self-certified group key generation for ad hoc clusters in wireless sensor networks. In *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.* IEEE, 2005.

[AQR07] O. Arazi, H. Qi, and D. Rose. A public key cryptographic method for denial of service mitigation in wireless sensor networks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, 2007.

[Ara99] Benjamin Arazi. Certification of dl/ec keys. Presented at IEEE P1363 group meeting, August 1998, 1999.

[ARMa] ARM Ltd. AMBA Open Specifications. `www.arm.com/products/solutions/axi_spec.html`.

[ARMb] ARM Ltd. ARM - Architecture. www.arm.com/products/CPUs/families/Cortex-M-Series.html.

[ARMc] ARM Ltd. ARM and Thumb2 instruction set - quick reference card. infocenter.arm.com/help/topic/com.arm.doc.qrc0001m/QRC0001_UAL.pdf.

[ARMd] ARM Ltd. ARM Cortex-M0 - technical reference manual. infocenter.arm.com/help/topic/com.arm.doc.ddi0432c/DDI0432C_cortex_m0_r0p0_trm.pdf.

[ARMe] ARM Ltd. ARM Thumb 16-bit instruction set - quick reference card. infocenter.arm.com/help/topic/com.arm.doc.qrc0006e/QRC0006_UAL16.pdf.

[ARMf] ARM Ltd. Building a Secure System using TrustZone Technology. infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[ARMg] ARM Ltd. Cortex-A5 technical reference manual - secure configuration register. infocenter.arm.com/help/topic/com.arm.doc.ddi0433a/CIHBJJCF.html.

[ARMh] ARM Ltd. Official documentation about ARM Cortex M0 processor. www.arm.com/products/processors/cortex-m/cortex-m0.php.

[ARMi] ARM Ltd. Official documentation about ARM TrustZone. www.arm.com/products/processors/technologies/trustzone.php.

[Atma] Atmel Corporation. Atmel AVR XMEGA - Architecture. www.atmel.com/products/AVR/default_xmega.asp.

[Atmb] Atmel Corporation. Atmel AVR XMEGA - documentation. www.atmel.com/products/AVR/xmega.asp?family_id=607&source=avr32.

[Atmc] Atmel Corporation. Atmel AVR XMEGA - how to reduce power consumption. www.atmel.com/dyn/resources/prod_documents/doc8267.pdf.

[Atmd] Atmel Corporation. Atmel AVR XMEGA - malual. www.atmel.com/dyn/resources/prod_documents/doc8077.pdf.

[Atme] Atmel Corporation. Atmel AVR XMEGA - power consumption. embedded-system.net/atmel-avr-xmega-microcontrollers.html.

[Atmf] Atmel Corporation. Atmel SecureAVR. www.atmel.com/dyn/products/devices.asp?family_id=662.

[Atmg] Atmel Corporation. Atmel SecureAVR - at90sc288144ru. www.atmel.com/dyn/products/product_card.asp?part_id=4141.

[Atmh] Atmel Corporation. Atmel SecureAVR overview. www.atmel.com/products/SecureAVR/.

[AUJP04] Sasikanth Avancha, Jeffrey L. Undercoffer, Anupam Joshi, and John Pinkston. *Security for Sensor Networks*. Kluwer Academic Publishers, 2004.

[AVRa] AVRfreaks - Bingo600. AVR XMEGA - description how to use AVR XMEGA toolchain. www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=86953.

[AVRb] AVRfreaks - Bingo600. AVR XMEGA - toolchain package for a Ubuntu 9.10, x86-64 machine. www.wrightflyer.co.uk/avr-gcc/avr-gcc-4.3.4-avrfreaks-09-mar-2010.deb.

[Ban08] Sukla Banerjee. Detection/removal of cooperative black and gray hole attack in mobile ad-hoc networks. In *WCECS'08: Proceedings of the World Congress on Engineering and Computer Science 2008*. International Association of Engineers, Newswood Limited, 2008.

[BBBD06] Alexander Becher, Er Becher, Zinaida Benenson, and Maximillian Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *International Conference on Security in Pervasive Computing (SPC)*, 2006.

[BCD$^+$05a] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. In *ACM Mobile Networks and Applications, Vol 10, No. 4*, 2005.

[BCD$^+$05b] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: an embedded multi-threaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 2005.

[BDSV$^+$98] Carlo Blundo, Alfredo De Santis, Ugo Vaccaro, Amir Herzberg, Shay Kutten, and Moti Yong. Perfectly secure key distribution for dynamic conferences. *Inf. Comput.*, 1998.

[BDV05] Levente Buttyán, László Dóra, and István Vajda. Statistical wormhole detection in sensor networks. *Security and Privacy in Ad-hoc and Sensor Networks*, 2005.

[Ber08] Daniel J. Bernstein. RSA signatures and Rabin-Williams signatures: the state of the art. 2008.

[BG06] Vijay Bhuse and Ajay Gupta. Anomaly intrusion detection in wireless sensor networks. *Journal on High Speed Netw.*, 2006.

[BGD05] Michael Brownfield, Yatharth Gupta, and Nathaniel Davis. Wireless sensor network denial of sleep attack. In *IEEE Information Assurance and Security Workshop (IASW)*, 2005.

[BGHL10] Levente Buttyán, Dennis Gessner, Alban Hessler, and Peter Langendörfer. Application of wireless sensor networks in critical infrastructure protection: Challenges and design options. *IEEE Wireless Communications Magazine, Special Issue on Security and Privacy in Emerging Wireless Networks*, 2010.

[BHSS03] B. Blum, T. He, S. Son, and J. Stankovic. Igf: A state-free robust communication protocol for wireless sensor network. Technical Report CS-2003-11, University of Virginia, 2003.

[BHUW09] Jens-Matthias Bohli, Alban Hessler, Osman Ugus, and Dirk Westhoff. Security enhanced multi-hop over the air reprogramming with fountain codes. In *LCN*. IEEE, 2009.

[BLM07] Chakib Bekara and Maryline Laurent-Maknavicius. A secure aggregation protocol for cluster-based wireless sensor networks with no requirements for trusted aggregator nodes. In *International Conference on Next Generation Mobile Applications, Services and Technologies*. IEEE, 2007.

[BLMB07] Chakib Bekara, Maryline Laurent-Maknavicius, and Kheira Bekara. Sapc: a secure aggregation protocol for cluster-based wireless sensor networks. In *International conference on Mobile ad-hoc and sensor networks (MSN)*. Springer, 2007.

[Blo85] R. Blom. An optimal class of symmetric key generation systems. In *Workshop on Advances in cryptology: theory and application of cryptographic techniques (EUROCRYPT)*. Springer, 1985.

[BMAAdG09] Zorana Banković, José M. Moya, Álvaro Araujo, and Juan-Mariano de Goyeneche. Intrusion detection in sensor networks using clustering and immune systems. In *Lecture Notes in Computer Science*, 2009.

[BRSS08] Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage. When good instructions go bad: generalizing return-oriented programming to RISC. In *Computer and Communications Security (CCS)*. ACM, 2008. Generalisation of (known) return-to-libc exploitation technique. Successfully and obviously breaks most if not all attestation techniques.

[BT03] Mathias Bohge and Wade Trappe. An authentication framework for hierarchical ad hoc sensor networks. In *Workshop on Wireless security (WiSe'03)*. ACM, 2003.

[BY03] Mihir Bellare and Bennet Yee. Forward-Security in Private-Key Cryptography. In *Topics in Cryptology - CT-RSA 2003*. Springer, 2003.

[CC07] Xiaomei Cao and Guihai Chen. ROSS: Resource oriented security solution for heterogeneous clustered sensor networks. *International Journal of Intelligent Control and Systems*, 2007.

[CCH07] M. Cagalj, S. Capkun, and J.-P. Hubaux. Wormhole-based antijamming techniques in sensor networks. *IEEE Transactions on Mobile Computing*, 2007.

[CCWW00] Mike Chen, Weidong Cui, Alec Woo, and Victor Wen. Security and deployment issues in a sensor network. 2000.

[CFPS09] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. On the difficulty of software-based attestation of embedded devices. In *Computer and Communications Security (CCS)*. ACM, 2009.

[Cod] CodeSourcery. Cross-compile toolchain. www.codesourcery.com/sgpp/lite/arm/portal/release1033.

[Cora] Atmel Corporation. picoPower Technology: Hammering Down Power Consumption - Maintaining Performance. www.atmel.com/ad/picopower/.

[Corb] Moteiv Corporation. TMote Sky Datasheep. http://www.bandwavetech.com/download/tmote-sky-datasheet.pdf.

[Corc] OAR Corporation. Rtems operating system. www.rtems.com/.

[CP03] Haowen Chan and Adrian Perrig. Security and privacy in sensor networks. *Computer*, 2003.

[CPG06] Garth V. Crosby, Niki Pissinou, and James Gadze. A framework for trust-based cluster head election in wireless sensor networks. In *Workshop on Dependability and Security in Sensor Networks and Systems (DSSNS'06)*. IEEE, 2006.

[CPS03] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 2003.

[CPS06] Haowen Chan, Adrian Perrig, and Dawn Song. Secure hierarchical in-network aggregation in sensor networks. In *Computer and Communications Security (CCS)*. ACM, 2006.

[Cri] Common Criteria. Common criteria - homepage. www.commoncriteriaportal.org/index.html.

[CRM$^+$07] Francisco J. Claudio, Rico Radeke, Dimitri Marandin, Petia Todorova, and Slobodanka Tomic. Performance study of reconfiguration algorithms in cluster-tree topologies for wireless sensor networks. In *Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2007. Part of PIRMS'07.

[Cro] Crossbow Technology Inc. MICAz - whireless measurement system. www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[CV95] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 1995.

[CWZG07] Haiguang Chen, Huafeng Wu, Xi Zhou, and Chuanshan Gao. Agent-based trust model in wireless sensor networks. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*. IEEE, 2007.

[cY04] Seyit Ahmet Çamtepe and Bülent Yener. Combinatorial design of key distribution mechanisms for wireless sensor networks. In *ESORICS*. Springer, 2004.

[cY05] Seyit Ahmet Çamtepe and Bülent Yener. Key distribution mechanisms for wireless sensor networks: a survey. Technical report, Rensselaer Polytechnic Institute, 2005.

[dAFN$^+$08] André L. L. de Aquino, Carlos M. S. Figueiredo, Eduardo F. Nakamura, Alejandro C. Frery, Antonio A. F. Loureiro, and Antônio Otávio Fernandes. Sensor stream reduction for clustered wireless sensor networks. In *ACM symposium on Applied computing (SAC)*. ACM, 2008.

[DCL04] Bruno Dutertre, Steven Cheung, and Joshua Levy. Lightweight key management in wireless sensor networks by leveraging initial trust. Technical Report SRI-SDL-04-02, System Design Laboratory, SRI International, 2004.

[DDH$^+$04] Wenliang Du, Jing Deng, Yunghsiang S. Han, Shigang Chen, and Pramod K. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. In *INFOCOM*, 2004.

[DDHV03] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Computer and Communications Security (CCS)*. ACM, 2003.

[DFN06] Wenliang Du, Lei Fang, and Peng Ning. Lad: localization anomaly detection for wireless sensor networks. *J. Parallel Distrib. Comput.*, 2006.

[DGV04] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *IEEE Workshop on Embedded Networked Sensors*, 2004.

[DGXC07] X. Du, M. Guizani, Y. Xiao, and H.-H. Chen. Two tier secure routing protocol for heterogeneous sensor networks. *IEEE Transactions on Wireless Communications*, 2007.

[DHM03] Jing Deng, Richard Han, and Shivakant Mishra. A performance evaluation of intrusion-tolerant routing in wireless sensor networks. In *International conference on Information processing in sensor networks*. Springer, 2003.

[DHM04] Jing Deng, Richard Han, and Shivakant Mishra. Intrusion tolerance and anti-traffic analysis strategies for wireless sensor networks. In *International Conference on Dependable Systems and Networks*. IEEE, 2004.

[DHM05] Jing Deng, Richard Han, and Shivakant Mishra. Defending against path-based dos attacks in wireless sensor networks. ACM, 2005. General Chair-Atluri, Vijay and Program Chair-Ning, Peng and Program Chair-Du, Wenliang.

[DHM06a] Jing Deng, Richard Han, and Shivakant Mishra. Insens: Intrusion-tolerant routing for wireless sensor networks. *Computer Communications*, 2006.

[DHM06b] Jing Deng, Richard Han, and Shivakant Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *International conference on Information processing in sensor networks (IPSN)*. ACM, 2006.

[DI] Jeff Dionne and Michael Durrant Arcturus Networks Inc. uclinux. www.uclinux.org/.

[DLL$^+$09] Dezun Dong, Mo Li, Yunhao Liu, , and Xiangke Liao. Wormcircle: Topological detection on wormholes in wireless sensor networks. In *International Conference on Parallel and Distributed Systems (ICPADS)*, 2009.

[Dou02] John R. Douceur. The sybil attack. In *IPTPS*. Springer, 2002. Revised Papers.

[DPMM03] Roberto Di Pietro, Luigi V. Mancini, and Alessandro Mei. Random key-assignment for secure wireless sensor networks. In *ACM workshop on Security of ad hoc and sensor networks (SASN)*. ACM, 2003.

[DQW03] Swades De, Chunming Qiao, and Hongyi Wu. Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks. *Computer Networks*, 2003.

[DS06] Murat Demirbas and Youngwhan Song. A rssi-based scheme for sybil attack detection in wireless sensor networks. In *Symposium on World of Wireless, Mobile and Multimedia Networks (WOWMIM'06)*. IEEE, 2006.

[dSMR$^+$05] Ana Paula R. da Silva, Marcelo H. T. Martins, Bruno P. S. Rocha, Antonio A. F. Loureiro, Linnyer B. Ruiz, and Hao Chi Wong. Decentralized intrusion detection in wireless sensor networks. In *International workshop on Quality of service & security in wireless and mobile networks (Q2SWinet'05)*. ACM, 2005.

[DSWC09] Huijuan Deng, Xingming Sun, Baowei Wang, and Yuanfu Cao. Selective forwarding attack detection using watermark in wsns. In *CCCM'09: International Colloquium on Computing, Communication, Control, and Management*, 2009.

[EAA$^+$06] Itamar Elhanany, Ortal Arazi, Benjamin Arazi, Derek Rose, and Senior Hairong Qi. Self-certified public key generation for resource-constrained sensor networks. Published as free pdf proceedings, 2006.

[EE01] Jeremy Elson and Deborah Estrin. Time synchronization for wireless sensor networks. In *International Parallel &amp; Distributed Processing Symposium (IPDPS'01)*. IEEE, 2001.

[EG02] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *Computer and Communications Security (CCS)*. ACM, 2002.

[Eur] European Space Agency (ESA). Website. www.esa.int.

[FB08] Felix Freiling and Zinaida Benenson. Attacker models for wireless sensor networks. summer school "protocols and security for wireless sensor actor networks", schloss dagstuhl. Slides of presentation, University of Mannheim, Germany, 2008.

[FC08] Aurélien Francillon and Claude Castelluccia. Code injection attacks on harvard-architecture devices. In *Computer and Communications Security (CCS)*. ACM, 2008.

[FG10] Dario Fiore and Rosario Gennaro. Making the diffie-hellman protocol identity-based. In *Topics in Cryptology - CT-RSA 2010*. Springer, 2010.

[FH09] Rainer Falk and Hans-Joachim Hof. Fighting insomnia: A secure wake-up scheme for wireless sensor networks. In *International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*. IEEE, 2009.

[Frea] Free Software Foundation, Inc. GDB - The GNU Project Debugger. `www.gnu.org/software/gdb/`.

[Freb] Free Software Foundation, Inc. GNU Lesser General Public License. `www.gnu.org/licenses/lgpl.html`.

[Gaia] Aeroflex Gaisler. GRFPU High-Performance Floating-Point Unit. `www.gaisler.com/cms/index.php?option=com_content&task=view&id=138&Itemid=54`.

[Gaib] Aeroflex Gaisler. LEON Bare-C Cross Compilation System (BCC). `www.gaisler.com/cms/index.php?option=com_content&task=view&id=147&Itemid=31`.

[Gaic] Aeroflex Gaisler. Leon2 - Architecture. `www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=52`.

[Gaid] Aeroflex Gaisler. Leon2 Processor. `www.gaisler.com/cms/index.php?option=com_content&task=section&id=4&Itemid=33`.

[Gaie] Aeroflex Gaisler. Leon2 Processor description. `vlsicad.eecs.umich.edu/BK/Slots/cache/www.gaisler.com/products/leon2/leon.html`.

[Gaif] Aeroflex Gaisler. Leon3 - Architecture. `www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53`.

[Gaig] Aeroflex Gaisler. Leon3 Processor. `www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53`.

[Gaih] Aeroflex Gaisler. LEON3FT-RTAX Fault-tolerant Processor . `www.gaisler.com/cms/index.php?option=com_content&task=view&id=196&Itemid=151`.

[GBS08] Saurabh Ganeriwal, Laura K. Balzano, and Mani B. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Trans. Sen. Netw.*, 2008.

[GDKP10] Thanassis Giannetsos, Tassos Dimitriou, Ioannis Krontiris, and Neeli R. Prasad. Arbitrary code injection through self-propagating worms in von neumann architecture devices. *The Computer Journal Advance Access*, 2010.

[GFJ$^+$09] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2009.

[GGSE01] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2001.

[GKS03] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys'03)*. ACM, 2003.

[Gmb] IHP GmbH. Ihp - innovations for high performance microelectronics - homepage. `www.ihp-microelectronics.com/`.

[GN08] Qijun Gu and Rizwan Noorani. Towards self-propagate mal-packets in sensor networks. In *Wireless network security (WiSec'08)*. ACM, 2008.

[Goo07] Travis Goodspeed. Msp430 buffer overflow exploit for wireless sensor nodes. http://travisgoodspeed.blogspot.com/2007/08/machine-code-injection-for-wireless.html, 2007.

[Goo08] Travis Goodspeed. Stack overflow exploits for msp430 wireless sensors over 802.15.4, 2008. In Texas Instruments Developer Conference.

[Goo09] Travis Goodspeed. Reversing and exploiting wireless sensors. Black Hat DC Briefings 2009, 2009. Work in progress.

[GPvS08] Saurabh Ganeriwal, Christina Pöpper, Srdjan Čapkun, and Mani B. Srivastava. Secure time synchronization in sensor networks. *ACM Trans. Inf. Syst. Secur.*, 2008.

[GPW$^+$04] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling C. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Cryptographic Hardware and Embedded Systems - CHES 2004*. Springer, 2004.

[Gra] Mentor Graphics. ModelSim SE. model.com/content/modelsim-se-high-performance-simulation-and-debug.

[GTS$^+$09] Saurabh Ganeriwal, Ilias Tsigkogiannis, Hohyun Shim, Vlassios Tsiatsis, Mani B. Srivastava, and Deepak Ganesan. Estimating clock uncertainty for efficient duty-cycling in sensor networks. *IEEE/ACM Trans. Netw.*, 2009.

[Gut01] Peter Gutmann. Data remanence in semiconductor devices. In *USENIX Security Symposium*. USENIX, 2001.

[GvHS] Saurabh Ganeriwal, Srdjan Čapkun, Chih C. Han, and Mani B. Srivastava. Secure time synchronization service for sensor networks.

[HBH05] Carl Hartung, James Balasalle, and Richard Han. Node compromise in sensornetworks: The need for secure systems. Technical report, Department of Computer Science University of Colorado, 2005.

[HCB00] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Hawaii International Conference on System Sciences (HICSS'00)*. IEEE, 2000.

[HCGD06] Song Han, Elizabeth Chang, Li Gao, and Tharam Dillon. Taxonomy of attacks on wireless sensor networks. In *European Conference on Computer Network Defence School of Computing (EC2ND)*. Springer, 2006.

[HCK$^+$03] Qiang Huang, Johnas Cukier, Hisashi Kobayashi, Bede Liu, and Jinyun Zhang. Fast authenticated key establishment protocols for self-organizing sensor networks. In *Proceedings of the Second ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*. ACM, 2003.

[HHC07] Meng-Yen Hsieh, Yueh-Min Huang, and Han-Chieh Chao. Adaptive security design with malicious node detection in cluster-based sensor networks. *Comput. Commun.*, 2007.

[HHJ09] Tran Hoang Hai, Eui-Nam Huh, and Minho Jo. A lightweight intrusion detection framework for wireless sensor networks. *Wireless Communications and Mobile Computing*, 2009.

[HIJM09] Z.M. Hanapi, M. Ismail, K. Jumari, and M. Mahdavi. Dynamic window secured implicit geographic forwarding routing for wireless sensor network. In *World Academy of Science, Engineering and Technology. Int. Conf. Wireless Commun. Sensor Network*, 2009.

[HKS$^+$05] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor networks. In *International conference on Mobile systems, applications, and services (MobiSys)*. ACM, 2005.

[HLK07] K.-S. Hung, K.-S. Lui, and Y.-K. Kwok. A trust-based geographical routing scheme in sensor networks. In *WCNC'07: Wireless Communications & Networking Conference*, 2007.

[HLV04] David Hwang, Bo-Cheng Lai, and Ingrid Verbauwhede. Energy-memory-security tradeoffs in distributed sensor networks. In *ADHOC-NOW*. Springer, 2004.

[HMORH06a] Md. Abdul Hamid, Md. Mamun-Or-Rashid, and Choong Seon Hong. Defense against laptop class attacker in wireless sensor network. In *International Conference of Advanced Communication Technology (ICACT'06)*, 2006.

[HMORH06b] Md. Abdul Hamid, Md. Mamun-Or-Rashid, and Choong Seon Hong. Routing security in sensor network: Hello flood attack and defense. In *International Conference on Next-Generation Wireless Systems (ICNEWS'06)*, 2006.

[HNLD08] Sangwon Hyun, Peng Ning, An Liu, and Wenliang Du. Seluge: Secure and dos-resistant code dissemination in wireless sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'08)*. IEEE, 2008.

[Hooa] Hoozi Resources Team. Advanced encryption standard (AES) implementation in C/C++ with comments. part 1: Encryption. `www.hoozi.com/post/829n1/advanced-encryption-standard-aes-implementation-in-c-c-with-comments-part-1-encryption`.

[Hoob] Hoozi Resources Team. Advanced encryption standard (AES) implementation in C/C++ with comments. part 2: Decryption. `www.hoozi.com/post/neqd5/advanced-encryption-standard-aes-implementation-in-c-c-with-comments-part-2-decryption`.

[Hooc] Hoozi Resources Team. Hoozi resources team - homepage. `www.hoozi.com/about`.

[Hood] Hoozi Resources Team. Secure hash algorithm (sha-1) reference implementation in C/C++. `www.hoozi.com/post/b3mf9/secure-hash-algorithm-sha-1-reference-implementation-in-c-c-with-comments`.

[HPJ03a] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[HPJ03b] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *Workshop on Wireless security (WiSe'03)*. ACM, 2003.

[HPP$^+$07] Parisa Haghani, Panos Papadimitratos, Marcin Poturalski, Karl Aberer, and Jean-Pierre Hubaux. Efficient and robust secure aggregation for sensor networks. In *Workshop on Secure Network Protocols*. IEEE, 2007.

[HSH$^+$08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*. USENIX Association, 2008.

[HSW$^+$00a] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ASPLOS-IX: International conference on Architectural support for programming languages and operating systems*. ACM, 2000. Appeared as ACM Operating Systems Review 34.5.

[HSW$^+$00b] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 2000.

[Hui] Jonathan Hui. The Trickle Algorithm. `http://www.cs.berkeley.edu/~jwhui/deluge/deluge-manual.pdf`.

[HUW11] Isabelle Hang, Markus Ullmann, and Christian Wieschebrink. Short paper: A new identity-based DH key-agreement protocol for wireless sensor networks based on the arazi-qi scheme. In *Conference on Wireless Network Security (WiSec)*. ACM, 2011.

[HZR09] Kevin Hoffman, David Zage, and Cristina N. Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys*, 2009.

[HZX08] Dai Hongjun, Jia Zhiping, and Dong Xiaona. An entropy-based trust modeling and evaluation for wireless sensor networks. In *2008 International Conference on Embedded Software and Systems*. IEEE, 2008.

[IEEa] IEEE. *1076/INT-1991 IEEE Standards Interpretations: IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual*.

[IEEb] IEEE. *IEEE 754: Standard for Binary Floating-Point Arithmetic.*

[IEEc] IEEE. *IEEE Std 1754-1994 IEEE Standard for a 32-bit Microprocessor Architecture - Description.*

[IEE06] IEEE. *IEEE Std 802.15.4-2006. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, 2006.

[Inca] SPARC International Inc. The SPARC architecture manual - version 8. www.sparc.org/standards/V8.pdf.

[Incb] Synopsys Inc. Company profile. www.synopsys.com/Company/AboutSynopsys/Pages/CompanyProfile.aspx.

[JA] Gilles Fayad Jerome Azema. M-Shield Mobile Security Technology: making wireless secure. focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf.

[JDUX09] Murtuza Jadliwala, Qi Duan, Shambhu Upadhyaya, and Jinhui Xu. Towards a theory for securing time synchronization in wireless sensor networks. In *Conference on Wireless network security (WiSec'09)*. ACM, 2009. General Chair-Basin, David and Program Chair-Capkun, Srdjan and Program Chair-Lee, Wenke.

[JIB07] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 2007.

[Kar05] Z. Karakehayov. Using REWARD to detect team black-hole attacks in wireless sensor networks. In *Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005.

[KBG+09] Ioannis Krontiris, Zinaida Benenson, Thanassis Giannetsos, Felix C. Freiling, and Tassos Dimitriou. Cooperative intrusion detection in wireless sensor networks. In *European Conference on Wireless Sensor Networks (EWSN'09)*. Springer, 2009.

[KDF07] I. Krontiris, T. Dimitriou, and F. C. Freiling. Towards intrusion detection in wireless sensor networks. In *European Wireless Conference*, 2007.

[KDGM07] Ioannis Krontiris, Tassos Dimitriou, Thanassis Giannetsos, and Marios Mpasoukos. Intrusion detection of sinkhole attacks in wireless sensor networks. In *ALGOSENSORS'07: Algorithmic Aspects of Wireless Sensor Networks, Third International Workshop*. Springer, 2007.

[KGD08a] Ioannis Krontiris, Thanassis Giannetsos, and Tassos Dimitriou. Launching a sinkhole attack in wireless sensor networks; the intruder side. In *WIMOB '08: International Conference on Wireless & Mobile Computing, Networking & Communication*. IEEE, 2008.

[KGD08b] Ioannis Krontiris, Thanassis Giannetsos, and Tassos Dimitriou. Lidea: a distributed lightweight intrusion detection architecture for sensor networks. In *International conference on Security and privacy in communication netowrks (SecureComm'08)*. ACM, 2008.

[KGO01] Christophe Mourtel Karine Gandolfi and Francis Olivier. Electromagnetic analysis: Concrete results. In *Proceedings of CHES01, volume 2162 of Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[KHB02] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wirel. Netw.*, 2002.

[Kho08] Mohamed Osama Khozium. Hello flood counter measure for wireless sensor networks. *IJCSS: International Journal of Computer Science and Security*, 2008.

[KK00] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Annual international conference on Mobile computing and networking (MobiCom)*. ACM, 2000.

[KLAG06] K.D. Kang, Ke Liu, and Nael Abu-Ghazaleh. Securing geographic routing in wireless sensor networks. In *Symposium on Information Assurance (SIA'06)*, 2006.

[KR05] Mark Krasniewski and Bryan Rabeler. Tibfit: Trust index based fault tolerance for arbitrary data faults in sensor networks. In *International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 2005.

[KSMS07] Sophia Kaplantzis, Alistair Shilton, Nallasamy Mani, and Y. Ahmet Sekercioglu. Detecting selective forwarding attacks in wireless sensor networks using support vector machines. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP'07)*, 2007.

[KV02] Richard A. Kemmerer and Giovanni Vigna. Intrusion detection: A brief history and overview (supplement to computer magazine). *Computer*, 2002.

[KW03] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *SNPA'03: Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*. IEEE Computer Society, 2003. General Chair-Jakobsson, Markus and General Chair-Perrig, Adrian.

[LB05] I. Vajda L. Buttyan, L. Dora. Statistical wormhole detection in sensor networks. In *ESAS'05: Second European Workshop on Security and Privacy in Ad Hoc and Sensor Networks*, 2005.

[LC06] Suk-Bok Lee and Yoon-Hwa Choi. A resilient packet-forwarding scheme against maliciously packet-dropping nodes in sensor networks. ACM, 2006.

[LCCK07] Sang Hoon Lee, Byong-Ha Cho, Lynn Choi, and Sun-Joong Kim. Event-driven power management for wireless sensor networks. In *Software Technologies for Embedded and Ubiquitous Systems*. Springer, 2007.

[LCEH03] Yee Wei Law, Ricardo Corin, Sandro Etalle, and Pieter H. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In *PWC*. Springer, 2003.

[LCH$^+$] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm. http://tools.ietf.org/rfc/rfc6206.txt.

[LGN06] Patrick E. Lanigan, Rajeev Gandhi, and Priya Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2006.

[LHdHH05] Y. Law, P. Hartel, J. den Hartog, and P. Havinga. Link-layer jamming attacks on s-mac. In *European Workshop on Wireless Sensor Networks (EWSN)*. IEEE, 2005.

[LN03a] Donggang Liu and Peng Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Network and Distributed Systems Security (NDSS)*. Internet Society, 2003.

[LN03b] Donggang Liu and Peng Ning. Establishing pairwise keys in distributed sensor networks. In *ACM Conference on Computer and Communications Security*. ACM, 2003.

[LN03c] Donggang Liu and Peng Ning. Location-based pairwise key establishments for static sensor networks. In *Workshop on Security of ad hoc and sensor networks (SASN)*. ACM, 2003.

[LN04] Donggang Liu and Peng Ning. Multilevel μ-tesla: Broadcast authentication for distributed sensor networks. *ACM Trans. Embed. Comput. Syst.*, 2004.

[LN08] An Liu and Peng Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. IEEE, 2008.

[LNLP06] C. E. Loo, M. Y. Ng, C. Leckie, and M. Palaniswami. Intrusion detection for routing attacks in sensor networks. *International Journal of Distributed Sensor Networks*, 2006.

[LP04] Loukas Lazos and Radha Poovendran. Serloc: secure range-independent localization for wireless sensor networks. In *ACM Workshop on Wireless security (WiSe)*. ACM, 2004.

[LRAFG10] Javier Lopez, Rodrigo Roman, Isaac Agudo, and Carmen Fernandez-Gago. Trust management systems for wireless sensor networks: Best practices. *Computer Communications*, 2010.

[LS05a] Jooyoung Lee and Douglas R. Stinson. A combinatorial approach to key predistribution for distributed sensor networks. In *IEEE Wireless Communications and Networking Conference*, 2005.

[LS05b] Jooyoung Lee and Douglas R. Stinson. Deterministic key predistribution schemes for distributed sensor networks. In *Selected Areas in Cryptography*. Springer, 2005.

[LTS09] Hong Luo, Jiaming Tao, and Yan Sun. Entropy-based trust management for data collection in wireless sensor networks. In *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2009.

[Lub02] Michael Luby. Lt codes. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2002.

[LvHD+05] Yee Wei Law, Lodewijk van Hoesel, Jeroen Doumen, Pieter Hartel, and Paul Havinga. Energy-efficient link-layer jamming attacks against wireless sensor network mac protocols. In *Workshop on Security of ad hoc and sensor networks (SASN)*. ACM, 2005.

[LWZZ08] Shaohe Lv, Xiaodong Wang, Xin Zhao, and Xingming Zhou. Detecting the sybil attack cooperatively in wireless sensor networks. In *International Conference on Computational Intelligence and Security (CIS'08)*. IEEE, 2008.

[MC09] Soo Young Moon and Tae Ho Cho. Intrusion detection scheme against sinkhole attacks in directed diffusion based sensor networks. *IJCSNS International Journal of Computer Science and Network Security*, 2009.

[MGD07] Ritesh Maheshwari, Jie Gao, and Samir R. Das. Detecting wormhole attacks in wireless networks using connectivity information. In *INFOCOM*. IEEE, 2007.

[MHU+09] K. Maier, A. Hessler, O. Ugus, J. Keller, and D. Westhoff. Multi-hop over-the-air reprogramming of wireless sensor networks using fuzzy control and fountain codes. In *Workshop on Self-Organising Wireless Sensor and Communication Networks*, 2009.

[Mil91] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 1991.

[MKSL04] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 2004.

[MLWSW07] Ye Ming Lu and Vincent W. S. Wong. An energy-efficient multipath routing protocol for wireless sensor networks. *Int. J. Commun. Syst.*, 2007.

[MMSK08] Andreas Meier, Mehul Motani, Hu Siquan, and Simon Künzli. Dimo: distributed node monitoring in wireless sensor networks. In *International symposium on Modeling, analysis and simulation of wireless and mobile systems (MSWiM)*. ACM, 2008.

[Moo05] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[MR04] V. Mhatre and C. Rosenberg. Homogeneous vs. heterogeneous sensor networks: A comparative study. In *ICC '04: Proceedings of IEEE International Conference on Communications*, 2004.

[MRS05] Michael Manzo, Tanya Roosta, and Shankar Sastry. Time synchronization attacks in sensor networks. ACM, 2005. General Chair-Atluri, Vijay and Program Chair-Ning, Peng and Program Chair-Du, Wenliang.

[MWS08] David J. Malan, Matt Welsh, and Michael D. Smith. Implementing public-key infrastructure for sensor networks. *ACM Trans. Sen. Netw.*, 2008.

[NLL06] E. C. H. Ngae, J. Liu, and M. R. Lyu. On the intruder detection for sinkhole attack in wireless sensor networks. In *ICC'06: Proceedings of IEEE International Conference on Communications*, 2006.

[NLL07] Edith C. H. Ngai, Jiangchuan Liu, and Michael R. Lyu. An efficient intruder detection algorithm against sinkhole attacks in wireless sensor networks. *Comput. Commun.*, 2007.

[NN06] Hoang Lan Nguyen and Uyen Trang Nguyen. Study of different types of attacks on multicast in mobile ad hoc networks. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL)*. IEEE Computer Society, 2006.

[NP03] R. Negi and A. Perrig. Jamming analysis of mac protocols. In *Carnegie Mellon Technical Memo*, 2003.

[NSSP04] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The Sybil attack in sensor networks: analysis & defenses. In *IPSN'04: Proceedings of the 3rd international symposium on Information processing in sensor networks*. ACM, 2004.

[NXPa] NXP. Announcement of NXP LPC11xx using a Cortex M0 processor. de.mouser.com/nxplpc1100mcus/.

[NXPb] NXP Semiconductors. Company profile. www.nxp.com/profile/.

[oB] University of Berkeley. TinyOS Documentation. http://www.tinyos.net/.

[ODL$^+$07] Leonardo B. Oliveira, Ricardo Dahab, Julio Lopez, Felipe Daguano, and Antonio A. F. Loureiro. Identity-based encryption for sensor networks. In *International Conference on Pervasive Computing and Communications Workshops (PERCOMW'07)*. IEEE, 2007.

[OM05] I. Onat and A. Miri. An intrusion detection system for wireless sensor networks. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications*, 2005.

[One96] Aleph One. Smashing the stack for fun and profit. *Phrack*, 1996.

[Ope] OpenCores.org. OpenCores - ethernet mac 10/100 - overview. www.opencores.org/project,ethmac.

[Osi07] Evgeny Osipov. tinyLUNAR: One-byte multihop communications through hybrid routing in wireless sensor networks. In *Next Generation Teletraffic and Wired/Wireless Advanced Networking*, 2007.

[oSN] National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publications. www.itl.nist.gov/fipspubs/.

[Ott] Daniel Otte. Crypto-avr-lib, a set of implementations of different cryptographic primitives. www.das-labor.org/wiki/AVR-Crypto-Lib/en.

[PC07] Niki Pissinou and Garth V. Crosby. Cluster-based reputation and trust for wireless sensor networks. In *2007 4th IEEE Consumer Communications and Networking Conference*. IEEE, 2007.

[PFVS08] Anthonis Papadimitriou, Fabrice Le Fessant, Aline Carneiro Viana, and Cigdem Sengul. Fighting sinkhole attacks in tree-based routing topologies. Technical report, INRIA, 2008. RR-6811.

[PFVS09] Anthonis Papadimitriou, Fabrice Le Fessant, Aline Carneiro Viana, and Cigdem Sengul. Cryptographic protocols to fight sinkhole attacks on tree-based routing in wireless sensor networks. In *Workshop on Secure Network Protocols (NPSEC'09)*, 2009. held in conjunction with ICNP'09.

[PGS09] M. Pugliese, A. Giani, and F. Santucci. Weak process models for attack detection in a clustered sensor network using mobile agents. In *International Conference on Sensor Systems and Software (S-CUBE'09)*, 2009.

[PHC04] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *International conference on Embedded networked sensor systems (SenSys)*. ACM, 2004.

[PLGP06] Bryan Parno, Mark Luk, Evan Gaustad, and Adrian Perrig. Secure sensor network routing: a clean-slate approach. In *ACM CoNEXT conference (CoNEXT'06)*. ACM, 2006.

[PS09] Dr G Padmavathi and Mrs D Shanmugapriya. A survey of attacks, security mechanisms and challenges in wireless sensor networks. *International Journal of Computer Science and Information Security*, 2009.

[PSW$^+$01] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. Spins: security protocols for sensor networks. In *International conference on Mobile computing and networking (MobiCom)*. ACM, 2001.

[PSW04] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, 2004.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Proceedings of E-smart 2001, volume 2140 of Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[Qua] Qualcomm. Qualcomm - SecureMSM. www.qctconnect.com/products/securemsm.html.

[Ray08] David Richard Raymond. Denial-of-sleep vulnerabilities and defenses in wireless sensor network mac protocols. Technical report, Virginia Polytechnic Institute and State University, 2008.

[RB06] Stanislav Rost and Hari Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *Sensor and Ad Hoc Communications and Networks (SECON)*, 2006.

[RC07] K. B. Rasmussen and S. Capkun. Implications of radio fingerprinting on the security of sensor networks. In *International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm'07)*, 2007.

[RCK$^+$05] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. In *International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 2005.

[Res] Gaisler Research. Leon2 Processor User's manual. www.arl.wustl.edu/~lockwood/class/cse465-s05/papers/leon2-1_0_23-xst.pdf.

[RFS$^+$03] S. Ramaswamy, H. Fu, M. Sreekantaradhya, J. Dixon, and K. Nygard. Prevention of cooperative black hole attack in wireless ad hoc networks. In *International Conference on Wireless Networks (ICWN)*. CSREA Press, 2003.

[RGCL09] He Ronghui, Ma Guoqing, Wang Chunlei, and Fang Lan. Detecting and locating wormhole attacks in wireless sensor networks using beacon nodes. In *WCSET'09: World Congress on Science, Engineering and Technology*, 2009.

[RLZ08] Kui Ren, Wenjing Lou, and Yanchao Zhang. Leds: Providing location-aware end-to-end data security in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 2008.

[RM07] D. Raymond and S. Midkiff. Clustered adaptive rate limiting: Defeating denial-of-sleep attacks in wireless sensor networks. In *AFCEA/IEEE Military Communication Conference*, 2007.

[RM08] David R. Raymond and Scott F. Midkiff. Denial-of-service in wireless sensor networks: Attacks and defenses. *IEEE Pervasive Computing*, 2008.

[RMBM06] D. Raymond, R. Marchany, M. Brownfield, and S. Midkiff. Effects of denial of sleep attacks on wireless sensor network mac protocols. In *IEEE Information Assurance and Security Workshop (IASW)*, 2006.

[RMBM09] D. Raymond, R. Marchany, M. Brownfield, and S. Midkiff. Effects of denial of sleep attacks on wireless sensor network mac protocols. *IEEE Transactions on Vehicular Technology*, 2009.

[RSCD08] S. D. Roy, S. A. Singh, Subhrabrata Choudhury, and Narayan C. Debnath. Countering sinkhole and black hole attacks on sensor networks using dynamic trust management. In *ISCC*. IEEE, 2008.

[RSS06] Tanya Roosta, Shiuhpyng Shieh, and Shankar Sastry. Taxonomy of security attacks in sensor networks and countermeasures. In *In The First IEEE International Conference on System Integration and Reliability Improvements*, 2006.

[RZL05] R. Roman, J. Zhou, , and J. Lopez. On the security of wireless sensor networks. In *Lecture Notes in Computer Science*, 2005.

[RZS$^+$08] Michele Rossi, Giovanni Zanca, Luca Stabellini, Riccardo Crepaldi, Albert F. Harris III, and Michele Zorzi. Synapse: A network reprogramming protocol for wireless sensor networks using fountain codes. In *SECON*. IEEE, 2008.

[SA08a] Devu Manikantan Shila and Tricha Anjali. A game theoretic approach to gray hole attacks in wireless mesh networks. In *IEEE Military Communications Conference*, 2008.

[SA08b] D.M. Shila and T. Anjali. Defending selective forwarding attacks in WMNs. In *IEEE International Conference on Electro/Information Technology (EIT 2008)*. IEEE, 2008.

[San07] David Sanchez. Secure, accurate and precise time synchronization for wireless sensor networks. In *Workshop on QoS and security for wireless and mobile networks (Q2SWinet'07)*. ACM, 2007.

[SBK05] Bharath Sundararaman, Ugo Buy, and Ajay D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 2005.

[SBS02] Eugene Shih, Paramvir Bahl, and Michael J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking, (MobiCom)*. ACM, 2002.

[SC09] Piotr Szczechowiak and Martin Collier. Tinyibe: Identity-based encryption for heterogeneous sensor networks. In *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 2009.

[SCHM08] Ricardo Simón Carbajo, Meriel Huggard, and Ciarán McGoldrick. An end-to-end routing protocol for peer-to-peer communication in wireless sensor networks. In *Workshop on Middleware for network eccentric and mobile applications (MiNEMA'08)*. ACM, 2008.

[SCKH05] Chien-Chung Su, Ko-Ming Chang, Yau-Hwang Kuo, and Mong-Fong Horng. The new intrusion prevention and detection approaches for clustering-based sensor networks. In *Wireless Communications and Networking Conference*, 2005.

[SCT04] Umesh Shankar, Monica Chew, and J. D. Tygar. Side effects are not sufficient to authenticate software. In *USENIX Security Symposium*. USENIX, 2004.

[SJL$^+$06] Riaz A. Shaikh, Hassan Jameel, Sungyoung Lee, Saeed Rajput, and Young J. Song. Trust management problem in distributed wireless sensor networks. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*. IEEE, 2006.

[Sko05] Sergei P. Skorobogatov. Semi-invasive attacks – a new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, 2005.

[SLL09] Riaz A. Shaikh, Young K. Lee, and Sungyoung Lee. Energy consumption analysis of reputation-based trust management schemes of wireless sensor networks. In *International Conference on Ubiquitous Information Management and Communication (ICUIMC'09)*. ACM, 2009.

[SLP$^+$06] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. SCUBA: Secure code update by attestation in sensor networks. In *Workshop on Wireless security (WiSe)*. ACM, 2006.

[SLS$^+$05] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 2005.

[SM06] Indranil Saha and Debapriyay Mukhopadhyay. Location verification based defense against sybil attack in sensor networks. In *International Conference on Distributed Computing and Networking (ICDCN'06)*. Springer, 2006.

[SM09] Indranil Saha and Debapriyay Mukhopadhyay. Security against sybil attack in wireless sensor network through location verification. In *International Conference on Distributed Computing and Networking (ICDCN'09)*. Springer, 2009.

[SMLB07] Mario Strasser, Andreas Meier, Koen Langendoen, and Philipp Blum. Dwarf: Delay-aware robust forwarding for energy-constrained wireless sensor networks. In *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2007.

[SNW06] Kun Sun, Peng Ning, and Cliff Wang. Tinysersync: secure and resilient time synchronization in wireless sensor networks. In *Computer and Communications Security (CCS)*. ACM, 2006.

[SPT$^+$09] Yannis Stelios, Nikos Papayanoulas, Panagiotis Trakadas, Sotiris Maniatis, Helen C. Leligou, and Theodore Zahariadis. A distributed energy-aware trust management system for secure routing in wireless sensor networks. In *Mobile Lightweight Wireless Systems*. Springer, 2009.

[SPvDK04] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep K. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 2004.

[SSW11] Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. Lightweight remote attestation using physical functions. In *Wireless Network Security (WiSec)*. ACM, 2011.

[Sta00] Mika Stahlberg. Radio jamming attacks against two popular mobile networks. In *In: Helsinki University of Technology Seminar on Network Security*, 2000.

[SWAG07] M. Sirivianos, D. Westhoff, F. Armknecht, and G. Girao. Non-manipulable aggregator node election protocols for wireless sensor networks. In *WiOpt'07: 5th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, 2007.

[SWC09] Kuo-Feng Ssu, Wei-Tong Wang, and Wen-Chung Chang. Detecting sybil attacks in wireless sensor networks using neighboring information. *Comput. Netw.*, 2009.

[SWNN] Andre Sieber, Karsten Walther, Stefan Nurnberger, and Jorg Nolte. Power Management in Reflex. http://www.betriebssysteme.org/Aktivitaeten/Treffen/2009-Nuernberg/Programm/docs/reflex_pm.pdf.

[SZC07] Hui Song, Sencun Zhu, and Guohong Cao. Attack-resilient time synchronization for wireless sensor networks. *Ad Hoc Networks*, 2007. Security Issues in Sensor and Ad Hoc Networks.

[TB05] Pim Tuyls and Lejla Batina. RFID-Tags for Anti-Counterfeiting. In *Proceedings of the Cryptographers' Track at the RSA Conference 2006 (CT-RSA'06)*. Springer, 2005.

[TCa] TU-Cottbus. Cocos Project. http://idun.informatik.tu-cottbus.de/cocos/cgi/cocos-trac.fcgi/wiki/WikiStart.

[TCb] TU-Cottbus. House Control Project. http://idun.informatik.tu-cottbus.de/reflex/cgi/reflex-trac.fcgi/wiki/HouseControl.

[TCc] TU-Cottbus. Reflex HowTo. https://www-docs.tu-cottbus.de/betriebssysteme/public/projekte/reflex/Release1.6/ReflexHowto.pdf.

[TCd] TU-Cottbus. Reflex Release 1.6 Subversion. http://idun.informatik.tu-cottbus.de/reflex/svn/tags/Release1.6.

[TCe] TU-Cottbus. Reflex Release 1.6 Tar.Gz. https://www-docs.tu-cottbus.de/betriebssysteme/public/projekte/reflex/Release1.6/Reflex_Release1.6.tar.gz.

[TCf] TU-Cottbus. Reflex Website. http://idun.informatik.tu-cottbus.de/reflex/cgi/reflex-trac.fcgi.

[TCg] TU-Cottbus. Reflex WhitePaper. https://www-docs.tu-cottbus.de/betriebssysteme/public/projekte/reflex/Release1.6/ReflexWhitePaper.pdf.

[TC05] Gilman Tolle and David Culler. Design of an application-cooperative management system for wireless sensor networks. In *European Workshop on Wireless Sensor Networks (EWSN)*, 2005.

[TCG02] Trusted Computing Platform Alliance (TCPA). *Main Specification*, 2002. Version 1.1b.

[TCG04] Trusted Computing Group (TCG). *TCG Architecture Overview*, 2004.

[TCG05a] Trusted Computing Group (TCG). *TPM Main Specification*, 2005.

[TCG05b] Trusted Computing Group (TCG). *TPM Main Specification, Version 1.2*, 2005.

[TDBH04] S. Tanachaiwiwat, P. Dave, R. Bhindwale, and A. Helmy. Location-centric isolation of misbehavior and trust routing in energy-constrained sensor networks. In *EWCN'04: IEEE Workshop on Energy-Efficient Wireless Communications and Networks, in conjunction with IEEE IPCCC*, 2004.

[Texa] Texas Instruments Inc. MSP430x5xx Family User's Guide. focus.ti.com/lit/ug/slau208f/slau208f.pdf.

[Texb] Texas Instruments Inc. TI MSP 430 Architecture. focus.ti.com/mcu/docs/mcuprodoverview.tsp?familyId=342&sectionId=95&tabId=140.

[Texc] Texas Instruments Inc. TI MSP 430 Architecture - MSP430x5xx family - Users Guide. focus.ti.com/lit/ug/slau208f/slau208f.pdf.

[TG08] Gelareh Taban and Virgil D. Gligor. Efficient handling of adversary attacks in aggregation applications. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2008.

[Tin07] TinyAODV. Tinyos source code repository. cvs.sourcefourge.net/viewcvs.py/tinyos/tinyos-1.x/contrib/hsn/, 2007.

[TL09] Liang Tang and QiaoLiang Li. S-spin: A provably secure routing protocol for wireless sensor networks. In *ICCSN'09: International Conference on Communication Software and Networks*, 2009.

[TMK+09] Panagiotis Trakadas, Sotiris Maniatis, Panagiotis Karkazis, Theodore Zahariadis, Helen Leligou, and Stamatis Voliotis. A novel flexible trust management system for heterogeneous wireless sensor networks. In *ISADS'09: The 9th International Symposium on Autonomous Decentralized Systems*, 2009.

[TP08] Patrick Tague and Radha Poovendran. Modeling node capture attacks in wireless sensor networks. invited paper. In *46th Annual Allerton Conference on Communication, Control, and Computing*, 2008.

[TSŠ+06] Pim Tuyls, Geert-Jan Schrijen, Boris Škorić, Jan van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In *Cryptographic Hardware and Embedded Systems Workshop*. Springer, 2006.

[TY94] J.D. Tygar and Bennet Yee. Dyad: a system using physically secure coprocessors. In *Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*. Interactive Multimedia Association, John F. Kennedy School of Government, 1994.

[Und] Steve Underwood. A port of the GNU tools to the texas instruments MSP430 microcontrollers. mspgcc4.sourceforge.net/.

[UoC] Boulder University of Colorado. MANTIS, official online-documentation. http://mantisos.org/documentation/api/html-unstable/index.html.

[UWB09] Osman Ugus, Dirk Westhoff, and Jens-Matthias Bohli. A ROM-friendly secure code update mechanism for WSNs using a stateful-verifier tau-time signature scheme. In *WISEC*. ACM, 2009. General Chair-Basin, David and Program Chair-Capkun, Srdjan and Program Chair-Lee, Wenke.

[vDL03] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 2003.

[vHH04] L. F. W. van Hoesel and P. J. M. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *International Workshop on Networked Sensing Systems (INSS)*. Society of Instrument and Control Engineers (SICE), 2004.

[vMKT06] B. Škorić, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied Physics*, 2006.

[Wag04] David Wagner. Resilient aggregation in sensor networks. In *SASN*. ACM, 2004.

[WB04] Weichao Wang and Bharat Bhargava. Visualization of wormholes in sensor networks. In *ACM Workshop on Wireless security (WiSe)*. ACM, 2004.

[WFSH06] Anthony D. Wood, Lei Fang, John A. Stankovic, and Tian He. Sigf: a family of configurable, secure routing protocols for wireless sensor networks. ACM, 2006.

[WKC+04] Ronald Watro, Derrick Kong, Sue F. Cuti, Charles Gardiner, Charles Lynn, and Peter Kruus. Tinypk: securing sensor networks with public key technology. In *SASN*. ACM, 2004.

[WLZC08] Mi Wen, Hui Li, Yan-Fei Zheng, and Ke-Fei Chen. Tdoa-based sybil attack detection scheme for wireless sensor networks. *Journal of Shanghai University (English Edition)*, 2008.

[WN07] K. Walther and J. Nolte. A flexible scheduling framework for deeply embedded systems. In *Proceedings of the 4th IEEE International Symposium on Embedded Computing*, 2007.

[WRLZ09] Q Wang, K. Ren, W. Lou, and Y Zhang. Dependable and secure sensor data storage with dynamic integrity assurance. In *IEEE INFOCOM 2009*, 2009.

[WS02] Anthony D. Wood and John A. Stankovic. Denial of service in sensor networks. *Computer*, 2002.

[WS04] Anthony D. Wood and John A. Stankovic. A taxonomy for denial-of-service attacks in wireless sensor networks. In *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, 2004.

[WSC10] Wei-Tong Wang, Kuo-Feng Ssu, and Wen-Chung Chang. Defending sybil attacks based on neighboring relations in wireless sensor networks. *Security and Communication Networks*, 2010.

[WSS03] Anthony D. Wood, John A. Stankovic, and Sang H. Son. Jam: A jammed-area mapping service for sensor networks. In *International Real-Time Systems Symposium (RTSS)*. IEEE, 2003.

[WYSC07] Jiangtao Wang, Geng Yang, Yuan Sun, and Shengshou Chen. Sybil attack detection based on rssi for wireless sensor network. In *WiCom'07: International Conference on Wireless Communications, Networking and Mobile Computing, 2007*, 2007.

[XMTZ06] Wenyuan Xu, Ke Ma, W. Trappe, and Yanyong Zhang. Jamming sensor networks: attack and defense strategies. *Network, IEEE*, 2006.

[XOL+07] Yurong Xu, Yi Ouyang, Zhengyi Le, James Ford, and Fillia Makedon. Analysis of range-free anchor-free localization in a WSN under wormhole attack. In *Symposium on Modeling, analysis, and simulation of wireless and mobile systems (MSWiM)*. ACM, 2007.

[XTZW05] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *International symposium on Mobile ad hoc networking and computing (MobiHoc)*. ACM, 2005.

[XWTZ04] Wenyuan Xu, Timothy Wood, Wade Trappe, and Yanyong Zhang. Channel surfing and spatial retreats: Defenses against wireless denial of service. In *ACM workshop on Wireless security (WiSe)*. ACM, 2004.

[XYG07] Bin Xiao, Bo Yu, and Chuanshan Gao. Chemas: Identify suspect nodes in selective forwarding attacks. *J. Parallel Distrib. Comput.*, 2007.

[Ya07] Ya. Time synchronization methods for wireless sensor networks: A survey. *Program. Comput. Softw.*, 2007.

[YHE02] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *International Conference on Computer Communications (INFOCOM)*. IEEE, 2002.

[YHE04] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 2004.

[YQF07] Xianglan Yin, Wangdong Qi, and Fei Fu. Asts: An agile secure time synchronization protocol for wireless sensor networks. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2007.

[YSF09] Hai Yan, Zhijie J. Shi, and Yunsi Fei. Efficient implementation of elliptic curve cryptography on dsp for underwater sensor networks. Electronically published in http://www.imec.be/odes/odes-7_proceedings.pdf, 2009.

[YT08] Zhenwei Yu and Jeffrey J. P. Tsai. A framework of machine learning based intrusion detection for wireless sensor networks. In *International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'08)*. IEEE, 2008.

[YWCR07] Geng Yang, Jiangtao Wang, Hongbing Cheng, and Chunming Rong. An identity-based encryption scheme for broadcasting. In *IFIP International Conference on Network and Parallel Computing Workshops (NPC'07)*. IEEE, 2007.

[YWL09] K.Q. Yan, S.C. Wang, and C.W. Liu. A hybrid intrusion detection system of cluster-based wireless sensor networks. In *International MultiConference of Engineers and Computer Scientists (IMECS'09)*, 2009.

[YWZC07] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2007.

[YWZC08] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. Sdap: A secure hop-by-hop data aggregation protocol for sensor networks. *ACM Trans. Inf. Syst. Secur.*, 2008.

[YX06] Bo Yu and Bin Xiao. Detecting selective forwarding attacks in wireless sensor networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[YYA02] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *MASCOTS '02: International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. IEEE, 2002.

[YYY$^+$05] Hao Yang, Fan Ye, Yuan Yuan, Songwu Lu, and William Arbaugh. Toward resilient security in wireless sensor networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2005.

[Zia08] Tanveer A. Zia. Reputation-based trust management in wireless sensor networks. In *2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. IEEE, 2008.

[ZLV$^+$09] Theodore Zahariadis, Helen C. Leligou, Stamatis Voliotis, Sotiris Maniatis, Panagiotis Trakadas, and Panagiotis Karkazis. An energy and trust-aware routing protocol for large wireless sensor networks. In *WSEAS international conference on Applied informatics and communications (AIC'09)*. World Scientific and Engineering Academy and Society (WSEAS), 2009.

[ZMB08a] Wassim Znaidi, Marine Minier, and Jean-Philippe Babau. Detecting wormhole attacks in wireless networks using local neighborhood information. In *PIMRC*. IEEE, 2008.

[ZMB08b] Wassim Znaidi, Marine Minier, and Jean-Philippe Babau. An ontology for attacks in wireless sensor networks. Technical Report 6704, INRIA: INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE, 2008. ISSN 0249-6399, ISRN INRIA/RR–6704–FR+ENG.

[ZSJ03] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Computer and Communications Security (CCS)*. ACM, 2003.