University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

# Optimization of Two-way Authentication Protocol in Internet of Things

*Martin Noack*
*Zurich, Switzerland*
*Student ID: 09722232*

Supervisor: C. Schmitt, T. Bocek
Date of Submission: August 21, 2014

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

# Abstract

In this thesis, a new security scheme for authentication in wireless sensor networks is presented. The proposed solution uses a three-way handshake and a pre-shared masterkey to provide direct two-way authentication between entities. To reduce the hardware requirements, elliptic curve cryptography is used for key generation, signatures, and message encryption. The resulting implementation is very fast while using so little resources that it can be deployed on constrained devices, such as TelosB motes, while still providing a high level of security and low energy consumption. It is written for TinyOS version 2.1.2 and fully supports data aggregation and TinyIPFIX.

# Zusammenfassung

In dieser Arbeit wird ein neues Konzept zur Sicherheit in Wireless Sensor Networks vorgestellt. Die entwickelte Lösung nutzt einen drei-Wege-Handschlag und einen vorher geteilten Schlüssel um direkte und zwei-Wege Authentifizierung zu bieten. Um die Anforderungen an Hardware zu senken, wird Elliptische-Kurven-Kryptographie genutzt für die Erzeugung von Schlüsseln, Signaturen und Verschlüsselung von Nachrichten. Die implementierte Lösung ist schnell, obwohl sie so wenig Ressourcen nutzt, dass die Anwendung auf sehr beschränkten Knoten, wie den TelosB, problemlos läuft. Trotzdem bietet sie ein hohes Mass an Sicherheit und benötigt nur wenig Energie. Die Anwendung ist für TinyOS der Version 2.1.2 geschrieben und unterstützt Datenaggregation sowie TinyIPFIX.

# Contents

# Chapter 1

# Introduction

With the steadily growing impact of the Internet on everyday tasks and increasing diversity in connected devices, the topic 'Internet of Things' (IoT) is today more relevant than ever. The IoT describes a scenario in which not only personal computers and smartphones are interconnected, but everyday things are transformed into smart objects that can communicate and react to their environment [56]. Cisco estimates that the Internet of Things will grow to almost 50 billion installed units until 2020 [35]. Some of the most prominent areas of application will include home automation, assisted living, e-health, and enhanced learning [4].
Currently, two distinct trends in the IoT development can be observed. On the one hand, the Smart Dust movement, which tries to leaver extremely cheap and small devices, such as RFID chips [17]. On the other hand, the Web of Things (WoT), which aims to connect more powerful embedded IoT devices via Web protocols instead of communication on lower network layers [32]. This thesis focuses on the area of Wireless Sensor Networks (WSN), which describes an even more specific subset of the IoT and often show traits from both of the above movements. Figure 1.1 illustrates the conceptual role of WSNs in the IoT.
In a WSN, sensors are small and resource constrained devices, usually more powerful than RFID chips, yet too constrained to communicate with the Internet without serious restrictions. Sensors are spatially distributed and monitor their environmental conditions, such as temperature, light, and humidity. They transmit the recorded data through a wireless network to a central location. In the context of the IoT, they might find applications for monitoring in smart homes [100] or healthcare [3]. As an example, one might consider a smart door lock that transmits its status to a user's smart-phone. The user may then access the sensor readings from anywhere in the world, checking for instance if she forgot to lock the apartment door in the morning, or if a burglary was attempted.

## 1.1 Problem Statement

In a distributed wireless network, security is an important factor. While the outside temperature might not be a secret worth keeping, consider the example of a smart door

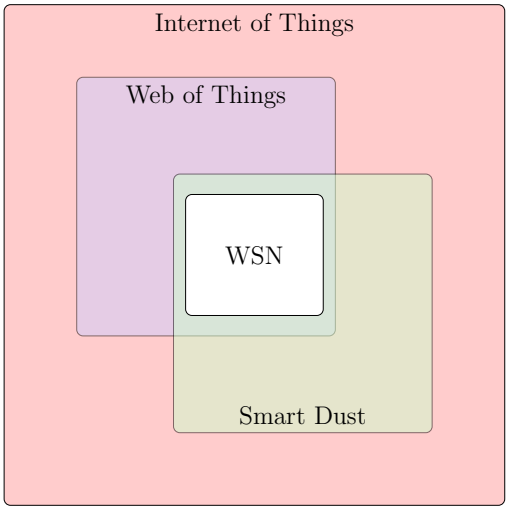



Figure 1.1: The relationship between IoT, WoT, WSN and Smart Dust.

lock. One can immediately identify two aspects of security that are very desirable in this scenario. Firstly, it must not be possible for just anyone, who is in wireless range, to pick up the fact that a door is unlocked. Eavesdropping is generally easy to implement, but very hard to detect in a distributed wireless environment [71]. Therefore, the access to information should be restricted and consequently the transmission of information should be secure. This problem can be solved by introducing encryption schemes for messages. Secondly, it is necessary to guarantee that a user is indeed who she claims to be and that she is authorized to receive the sensitive information. This can be achieved by implementing authentication and authorization mechanisms, effectively identifying a user and verifying her permission for access.
With the aforementioned use cases, it becomes apparent that end-to-end security is very desirable for applications in the IoT. Unfortunately, transmitted data is often unprotected in current implementations [91]. While there exist many established security solutions for generic computer systems on the Internet, one of the most notable characteristics of nodes in a WSN is the extreme constraint on resources. As computers become more and more powerful, sensor nodes, on the other hand, are still very limited in both memory and computational power (in the testbed of TelosB motes: 10 KB of RAM and 48 KB of ROM [81]). This is partially due to the desired very low energy consumption, as most WSN nodes run on battery power and should be operational for longer time periods without maintenance [62]. But most importantly, WSN nodes are supposed to be very small and cheap [22, 43].

In conclusion, the two main problems, that this thesis is aimed at, are: (1) to devise a handshake that provides the required mutual authentication and (2) to establish a secure communication channel, both for very resource constrained devices. Additionally, the solution should be compatible to existing implementations, such as aggregation support and IPFIX.

A more extensive list of goals and parameters for this thesis is the following:

1. The general goal is to improve existing solutions for security in resource constrained devices, namely TinySAM [66] and TinyDTLS [57].

2. The main contribution is the design of a two-way authenticated security solution that is suited for the constrained nodes in a WSN and provides support for existing algorithms, such as TinyIPFIX. Thus, a three-way handshake for two-way authentication between sensor nodes and the establishment of a secure communication channel is proposed and implemented.

3. Elliptic Curve Cryptography (ECC) should be used for key generation, signatures and encryption, to minimize the impact on memory consumption and computational capacities. With this approach, similar security levels as within RSA (Rivest Shamir Adleman) can be achieved, but with a significantly smaller key size. For further explanations see Section 2.5.

4. It has to be assumed that devices do not posses special hardware support (such as trusted platform modules).

5. Further, the solution should not rely on additional infrastructure (such as certificate authorities).

6. Given the basic scenario of spatially distributed sensor nodes which convey sensitive data with multi-hop routing, the solution should provide complete end-to-end security.

## 1.2 Thesis Outline

This thesis is conceptually structured as follows: First, the theoretical background for sensor nodes and their role in WSNs is examined in Chapter 2. This is accompanied by an in-depth analysis of potential threats and security fundamentals. In particular, elliptic curve cryptography is introduced and thoroughly explained, as it is an essential part of the security design in this thesis. Following up on the theoretical background, some alternative and related approaches are presented in Chapter 3, which use some of the concepts that apply as well to the proposed solution. Then the design of the two-way authentication handshake is outlined in Chapter 4, with a particular focus on th security considerations and goals in the subsequent protocol design. The resulting implementation of those concepts is described in Chapter 5 and includes an overview over TinyOS programming principles that were used for the implementation on motes and the subsequent architecture of the solution. In Chapter 6, the evaluation of the implementation is performed, which gives an overview over memory consumption, execution times, energy requirements, and the resulting security level. Further, the limitations of the current solution are presented. The thesis ends with a short summary and conclusion in Chapter 7.

# Chapter 2

# Background

This chapter contains an overview of the concept of sensor nodes, their capabilities, and a corresponding classification according to the Internet Engineering Task Force (IETF). The different roles that can be assigned to sensor nodes within the WSN will be outlined, further the most common threats to such a network are analyzed.

## 2.1 Sensor Nodes

In the context of WSNs, single sensor nodes are often called *motes* [81]. They include a processor, wireless radio, and often various sensors. Characteristically, motes collect environmental information and transmit those via radio link to a central entity, the so called *data sink*, or in short *sink*. Additionally, they are often equipped with sizable antennas to cover a longer distance, and a battery slot for the energy source. Figure 2.1 shows a IRIS and a TelosB mote, equipped both of the above.

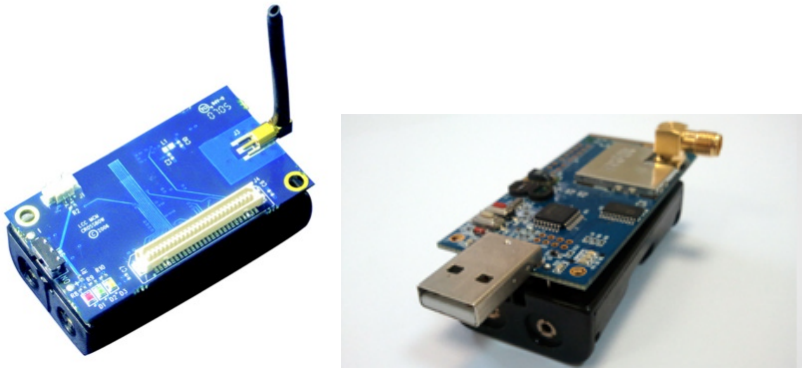


Figure 2.1: IRIS and TelosB CM5000-SMA motes.

In practice, motes come in various flavors. Physical size and computational power are highly depending on the target application. Mote sizes range from the proportions of

a shoebox, for example weather stations, to microscopically small particles in military applications, famously called *smart dust* [86, 104]. The deployment of motes consequently differs as well. Some motes may be deployed at random (for instance dropped from an airplane), or installed deliberately at a specific location. Table 2.1 gives an overview of commonly used WSN motes and their technical specifications to show the diversity in resources between different hardware platforms.

| **Mote** | **RAM [KB]** | **Flash [KB]** | **CPU [MHz]** | **Bus [bit]** | **Energy Supply** |
|---|---|---|---|---|---|
| TelosB | 10 | 48 | 4-8 | 16 | 2x AA battery / external |
| MicaZ / Mica2 | 4 | 128 | 8 | 8 | 2x AA battery |
| IRIS | 8 | 128 | 8 | 8 | 2x AA battery |
| OPAL | 52 | 256 | 96 | 32 | 3x AA battery / external |
| Rene | 0.5 | 8 | 4 | 8 | 1x button cell |

Table 2.1: List of common mote platforms and their hardware specs [51, 81, 50, 24].

In contrast to this broad spectrum of inhomogeneous hardware platforms, in the beginning of WSN research, it was often assumed that deployed devices are not only physically indistinguishable from each other, but even programmed identical [2]. However, in more recent applications, devices usually get assigned a unique ID and address in the address space, in order to uniquely identify and control them. As it is hence possible to combine various devices of different resource levels, it is not uncommon to select and program different motes specifically with certain roles in mind. A typical task for computationally more powerful devices is to collect, process, and route data from simpler sensor nodes, that are incapable of advanced computations on their own [86]. A commonly applied use-case is to aggregate data from several data collectors in one additional node to decrease to overall overhead in data transmission to the sink [58, 20].

### 2.1.1 Classification

To distinguish between different sizes and power levels for sensor nodes, the Internet Engineering Task Force (IETF) has defined classes of resource constrained devices [15]. The IETF has coined the term *constrained node* for them, to set the nodes apart from common devices, like laptops or smart-phones, in order to underline the contrast between typical characteristics of sensor nodes and expectations towards common Internet nodes. In particular, the IETF outlines three classes, which are summarized in Table 2.2. According to the definition does 1 Kibibyte (KiB) equal 1024 byte, whereas 1 Kilobyte (KB) equals 1000 byte.

Note that the given numbers are not necessarily representing strict rules, but may rather be seen as guidelines for classification. Classes were originally defined according to capabilities of devices, with a focus on network communications. Typically, the ability to run a fully compatible IP stack requires a certain amount of resources, which cannot be

| Class | Data size (RAM) | Code size (ROM) |
|---|---|---|
| 0 | ≪10 KiB | ≪100 KiB |
| 1 | ∼10 KiB | ∼100 KiB |
| 2 | ∼50 KiB | ∼250 KiB |

Table 2.2: IETF defined classes of constrained devices [15].

taken for granted with constrained nodes. The classification, therefore, corresponds to the amount of available resources, that in turn enable a certain subset of capabilities.

### Class 0 Devices

Devices of this class are extremely resource constrained in terms of both memory and processing power. They usually cannot communicate directly with the Internet in a secure manner and often use proxies, servers, or a gateway to establish communications. Class 0 devices cannot run an IP-stack as it is defined in the Requests for Comments (RFC) documents of the IETF. In practice, Class 0 devices are mostly preconfigured and preprogrammed and do not provide any form of security functions. Motes, as they are described in Section 2.1, often fall into this category. Especially the Rene mote stands out, which was used in smart dust research, and is extremely constrained with only 0.5 KB of RAM, even for class 0 devices [104].

### Class 1 Devices

Those devices are more powerful than Class 0 devices, but still provide only very limited code space (ROM) and processing power. They require lightweight protocol stacks for communication, which are specifically designed for constrained devices (one example is the Constrained Application Protocol (CoAP)). They can, however, communicate directly with the Internet and are powerful enough to provide support for basic security functions. Examples for this class are TelosB motes.

### Class 2 Devices

This is the class of the most powerful, but still constrained devices. Class 2 devices provide almost the same capabilities as laptops or smart-phones in terms of available protocol stacks and security functions. However, the devices may still benefit from lightweight and energy efficient implementations. The Berkeley OPAL mote is such a device. There exist constrained nodes which are more powerful than Class 2 devices, but those are likely to handle common protocol implementations easily and are, thus, not relevant for this classification.

## 2.2 WSN Components

As many authors note, it is often beneficial to assign different roles to WSN motes, most often to provide aggregation of collected environmental data [58, 20, 52, 70]. The idea is to shift from an address driven routing, where data is sent on the shortest path to the end-node, to a data driven routing, where data streams are combined in a meaningful way on the path to the destination. Min et al. have shown that the overall energy requirement for independent transmission of individual measurements towards a central entity is a lot higher than for aggregation and transmission of a combined data set once over the longer distance [70]. To visualize the idea, imagine a scenario where a user is interested in receiving measurements from all sensor nodes, but only to check if the average value has changed. For example the room temperature could be measured with several sensors in a single room, but the A/C control should only interfere if the average temperature has changed significantly. Therefore, an additional mote could compute the average of all sensor nodes in the room and forward only the aggregated average value, instead of every sensor transmitting its own values to the sink. This will help to save energy and transmission overhead to the data sink. However, it should be noted that an aggregation function can only be applied to meaningful data in order to produce meaningful results. In other words: it must be known what the data values represent. It would be meaningless to add up room temperatures for climate control, or to compute averages of patient heart rates. So in order to determine the appropriate aggregation function, it is required to know the meaning of the values.
Following those considerations, motes with different roles are deployed. Those are for one the *Collector* nodes, that collect data, and on the other side the *Aggregator* nodes, that aggregate data. A typical topology can be seen in Figure 2.2. In this scenario, two Aggregators are receiving data streams from two and three Collectors. They compute the aggregate of all values and signal the result to the Gateway, which is connected to the Internet. Additionally, three collectors transmit data directly to the Gateway.

### Collector Nodes

Collectors are limited to collecting and transmitting environmental data. They do not execute any preprocessing tasks on collected data, they send only raw information. Those could include, for example, humidity, temperature, light, and voltage. Measurements are executed periodically and immediately followed by data transmission to the sink.

### Aggregator Nodes

The Aggregators in this thesis differ from other implementations in a way that they do not collect sensor data on their own, they are used exclusively to process messages and to signal the results. This is not a general requirement, but applied in the implementation for this thesis, as TelosB motes do not offer enough memory and computational capacities for further tasks, like reading sensor values. The current implementations offers the option to change the *degree of aggregation* (doa) at runtime, meaning the number of transmissions

that are combined into one aggregate.
Aggregation can take two forms, both of which are supported in the current implementation. Firstly, it is possible to aggregate individual messages in one combined message, while the original message contents are preserved. This may be useful, for instance, when several spatially close motes need to transmit data to a common, but distant, sink. Introducing an Aggregator in close proximity to the Collectors, in this scenario, reduces the total energy consumption in the system, as multiple Collectors now have to transmit their data over a shorter distance than before, reducing the combined energy consumption. The long-distance transmission is executed only $\lceil \frac{n}{doa} \rceil$ times by the Aggregator, instead of $n$ times by Collectors, for $n$ Collectors.
Secondly, it is possible to apply an appropriate preprocessing function to the aggregated messages, e.g. to compute the average temperature from different Collectors. Aggregators wait until they have received the set amount of signals, then apply the aggregate functions to data values and transmit the results on completion.

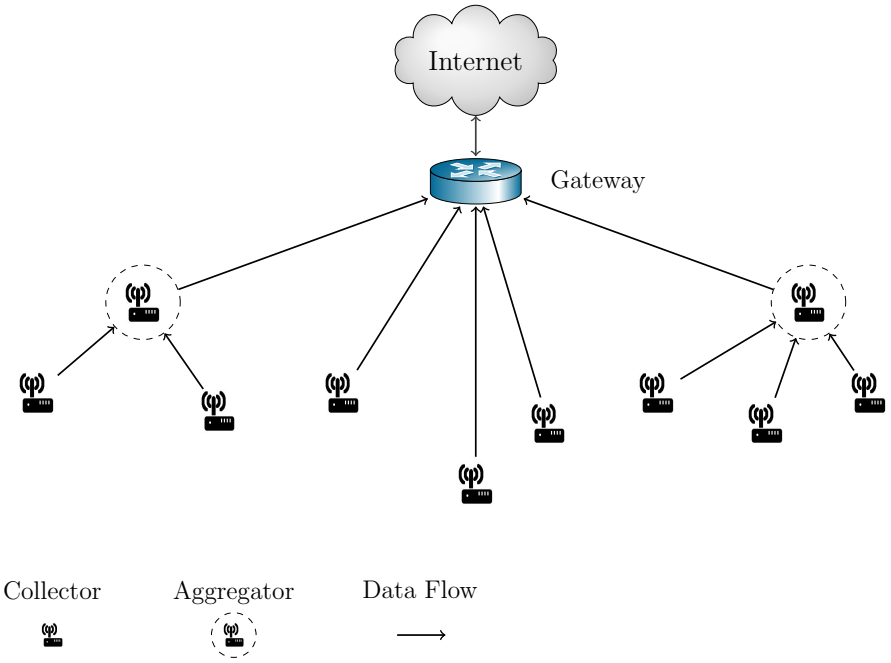



Figure 2.2: Typical WSN topology with Collectors and Aggregators.

### Gateway

The Gateway takes a special role in a WSN. It connects the local WSN to the Internet and routes data between both networks. Usually this includes a bridge between the IEEE 802.15.4 wireless personal area network (WPAN), used in WSNs, and the Internet. Commonly, this bridge is constructed as a combination of a sensor node (called base station), plus a server that is connected to the Internet. The resulting complex is referred to as *Gateway* and illustrated in Figure 2.3. The Gateway generally only forwards messages and does not perform data preprocessing. However, individual steps in data forwarding, such as the de-encapsulation of headers, cannot be attributed to single Gateway component. Thus, it is usually addressed as a single unit.

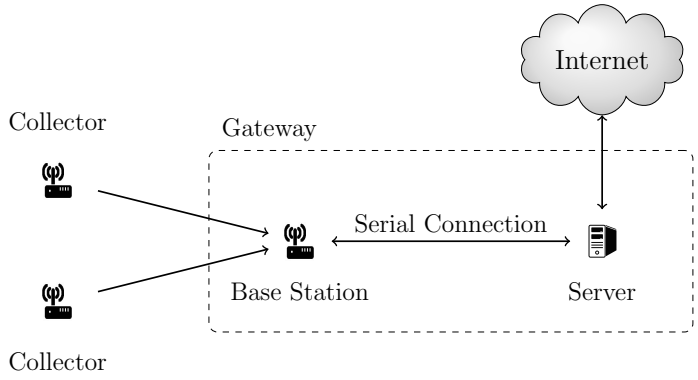



Figure 2.3: Gateway components and message flows.

## 2.3 Communication Standards

WSNs communicate over wireless channels, which may be configured in different ways. To advance interoperability and heterogeneousness of WSN, the Institute of Electrical and Electronics Engineers (IEEE) has defined the widely used IEEE 802.15.4 standard [96]. The standard specifies the physical layer and media access control for low-rate WPANs (LR-WPANs). It can be used as a foundation for upper layers and provides low-cost and low-speed ubiquitous communication between devices.
Based in the IEEE specifications, the IETF has devised a standard to enable end-to-end IPv6 networking in low-power WPANs (LowPANs) [59]. The result is called 6LowPAN (IPv6 over LowPAN) and defines the network layer specifications for the IPv6 communications in WSNs, built on top of IEEE 802.15.4 [72]. 6LowPAN defines the frame format for transmission of IPv6 packets, as well as the formation of IPv6 link-local addresses and statelessly autoconfigured addresses. However, the size of IPv6 packets is much larger than the biggest supported frames in IEEE 802.15.4: The IPv6 specifications require a Minimum Transmission Unit (MTU) of at least 1280 byte, IEEE 802.15.4 packets provide a maximum payload of 102 byte on the MAC layer (127 byte total frame size, minus 23 byte MAC header and 2 byte MAC footer). This issue is solved by introducing an adaption layer in the network stack, that takes over fragmentation and reassembly of IPv6 packets into IEEE 802.15.4 messages. RFC 4944 defines the technical details, such as Header-Compression and Address-Mapping [72]. In this thesis, the Berkeley Low Power IP (BLIP) stack is used as a 6LowPAN implementation, since it is the de-facto standard for TinyOS 2.x [113].
For routing in the IoT, the IETF has suggested the Routing Protocol for Low Power and Lossy Networks (RPL) [110]. RPL is a dynamic routing protocol and based on distance vector calculations. Its purpose is to provide efficient routing for three different traffic patterns in low power lossy networks (LLNs): Multipoint-to-point, point-to-multipoint, and point-to-point. RPL is specifically suited for multipoint-to-point patterns, like they are applied in this thesis: It creates a destination oriented, directed, acyclic graph (DODAG) for routing, which directs traffic towards one single end-node (i.e. the sink). Every node periodically informs the network about its position in the graph, allowing other nodes to determine the optimal next hop towards the sink with distance vector calculations.
In this thesis, those technologies were applied collectively, resulting in a network stack as it is shown in Figure 2.4. On the bottom, the IEEE 802.15.4 standard handles transmis-

sions on the physical (PHY) and data link (MAC) layer. On top of them is the network layer, which is handled by the BLIP stack and enables IPv6 communication with the help of LowPAN adaption from the data link layer. In order to keep the network stack as close as possible to the commonly used ISO/OSI model [48], the LowPAN adaption is not formally introduced as unique layer, but merely a component: It provides adaption for IPv6 communications to the actual abstraction layer. Further on top is the transport layer that handles UDP and ICMP packets and can be used by applications from the top most application layer.

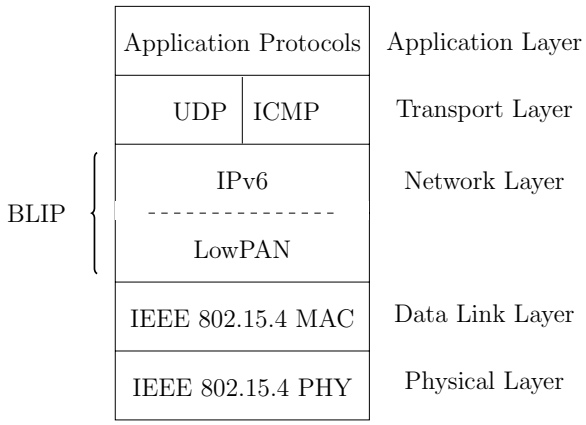



Figure 2.4: Network stack as used in this thesis.

## 2.4 Threats

Characteristically, Wireless Sensor Networks transmit messages over radio, using simple but unreliable protocols, like UDP. The security issues and threats to WSNs are, thus, closely related to those of other network types, like wireless ad hoc networks where devices do not rely on any pre-existing infrastructure but autonomously set it up on their own. However, a very specific characteristic of WSNs is the strictly defined flow of information within the network. While a wireless ad hoc network or peer-to-peer network is mostly self-organizing and enables bi-directional communications [79], a WSN typically shows a uni-directional message flow towards a pre-determined sink [86, 62]. Communications between individual sensors are usually not required beyond multi-hop forwarding or the exchange of routing information. The sole exception being Aggregator nodes, which in turn act as new data sink for their Collectors. This leads to a slight shift in priorities in security aspects. There exist quite exhaustive surveys and considerations regarding security challenges in WSNs [80, 78, 42, 103, 67], which will be examined in the following sections. The proposed solution will be evaluated against those threads in Section 6.4. Given the wide list of potential attack forms, it is obvious that many different aspects of security need to be considered in a practical solution. Table 2.3 summarizes the listed attacks and common defenses in the literature.

| Form | Attack | | Possible defense |
|---|---|---|---|
| Active | DoS | Physical Layer | Frequency hopping |
| | | Data Link Layer | Error correction codes |
| | | Network Layer | Egress filtering |
| | | Transport Layer | Authentication |
| | Sybil | Fabricated Identity | authentication / encryption |
| | | Stolen Identity | authentication / encryption |
| | Sinkhole | | Verify route quality |
| | Hello Flood | | Identity verification / client puzzles |
| | MITM | | Mutual authentication / encryption |
| | Physical | JTAG | Disable JTAG |
| | | BSL | Set password |
| | | Sensors | - |
| Passive | Eavesdropping | | Encryption |
| | Traffic Analysis | | Disguised routing |

Table 2.3: Attacks, their threat levels and proposed defense [80, 78, 42, 103, 67].

### 2.4.1 Active Attacks

The IETF categorizes *active attacks* as attempts to alter system resources or to affect their operations [92]. In essence, active attacks are executed with the intent to influence the system behavior, either by introducing new system resources or by altering existing ones, in order to gain access to otherwise restricted information.

#### Denial of Service (DoS)

A DoS attack is executed to render the target system inaccessible towards other users. This goal is achieved by either slowing down the target system or by preventing any access at all [92]. In WSNs, the obvious point of attack for a DoS is the wireless transmission medium. In general, there exist possible attacks on several different layers [78, 103]. The examples matching the applied network stack (cf. Section 2.3) are listed in the following paragraphs:

- Physical Layer

  **Jamming** the medium by deliberately broadcasting noise on a certain frequency or channel. Legitimate nodes may not be able to pick up the signal between all the noise.

- Data Link Layer

  **Collisions** can be caused by transmitting packets on the same frequency simultaneously with the sending unit. Packet collisions might introduce discrepancies in the data portions of a message and result in a checksum mismatch at the receiver side. In attacks, the adversary often tries to cause collisions for specific packets, such as ACK messages. If there is no limit to the (re)send attempts for nodes and the attacker causes collisions between ACK messages, nodes might try to send messages over and over again until their energy is depleted. This is called **Exhaustion**.

- Network Layer

  **Misdirection or spoofing of routing information** can lead to a disruptions in the network. Attackers might introduce routing cycles or increase the latency to extreme levels by artificially altering the routing path.

- Transport Layer

  **Flooding** is an attack that applies to protocols which include state information. By repeatedly starting connections, the target is forced to keep state information for every connection attempt
  **De-synchronization** denotes the disruption of an established communication channel. For instance by spoofing messages and causing retransmissions, a sender might be lured into the assumptions that is has to recover from errors that never existed.

Defending against DoS attacks that are executed with the intent to destroy network functionality is very difficult. Some of the potential counter measures include frequency hopping (against jamming), error-correcting codes (against collisions), egress filtering (against spoofed routing tables) and authentication (against de-synchronization) [42].

### Sybil Attack

In a Sybil attack, a malicious node presents itself to the network under multiple identities. This is, for instance, dangerous to distributed storage schemes [103]. In a scenario where data storage relies on replication, a Sybil node may identify itself as a valid storage entity, but in fact destroy all its data, essentially undermining the redundant system. Newsome et al. have created a detailed taxonomy of Sybil attacks on WSNs [74]. Most importantly, they decide between two types of Sybil attack executions:

- **Fabricated Identities**
  A Sybil node can simply create a new identity. If there is no mechanism in place to check the validity of identities, this behavior can easily go undetected. For instance, in a case where the IPv6 suffix is used to uniquely identify nodes, a Sybil node only needs to assign itself a valid IPv6 address with an available suffix.

- **Stolen Identities**
  If there exists an identification protocol, or even just a restricted address space to avoid the addition of new nodes, the Sybil node has to steal the identity of

an existing node. This sort of attack is often accompanied by the destruction or temporary deactivation of the original node.

To counter Sybil attacks, authentication and encryption can be used [78]. Even if the Sybil node can steal an identity and stay undetected, it has no immediate way of accessing the corresponding public / private key pair. In particular, Newsome et al. propose either *direct validation* or *indirect validation* as countermeasures to a Sybil attack. In the first, a node can directly test the identity of its opponent, in the latter a trusted node may vouch for others [74].

#### Sinkhole Attack

Sinkhole attacks are executed by forging routing information with the goal to make a compromised node more attractive to others [103, 78]. Consequently, most of the traffic is then routed through the attacking node, which can either compromise or interrupt data streams (often called *selective forwarding* [53]). A sinkhole attack can be countered by protocols that verify route quality, for instance by replying with acknowledgement and latency information [53].

#### Hello Flood Attack

Some routing protocols rely on the broadcast of hello messages to announce a node's presence to its neighbors. An attacker might use this to broadcast a strong signal over a long distance, reaching nodes that are themselves out of transmission range from the attacker. Yet, the victims will now assume that they are neighbors to the adversary and try to reach them. For this to work, attackers do not even need to create new traffic, they can just replay existing hello messages.
Countermeasures to hello flood attacks include secret-based multi-hop forwarding, identity verification protocols or client-puzzles to establish a bidirectional link [94].

#### Man-in-the-middle Attack

A malicious attacker can intercept messages between two legitimate peers and establish an independent connections to each peer. Peers then assume that they are talking to each other, while in reality, they communicate with the attacker, who only relays the message contents. The attacker has, full access to all messages between peers. He can read, alter, or destroy messages at will. Countermeasures include mutual identity authentication and message encryption.

#### Physical Attack

This completely different type of attacks is highly relevant to WSNs. Instead of exploiting bugs or software weaknesses, an attacker can often gain physical access to the whole sensor

mote, due to the spatial distribution of nodes. Becher et al. have developed a design space of physical attacks, from which one can derive the most relevant attack points [9]:

- **Joint Test Action Group (JTAG) Module**

  The so-called Test Access Ports (TAP) on the hardware platform are intended for developers to collect debugging information from the running program by using a JTAG module to read the debugging output. An attacker can use the TAP not only to get access to the microcontroller via debugging interfaces, but even to read the complete memory contents from the mote. This includes the deployed program in ROM and potentially secrets and key pairs in RAM.

- **Bootstrap Loader (BSL)**

  The typical way of programming a mote is over the BSL through the serial port. Physical access to the mote allows the attacker to connect over this port and to access RAM and ROM independently from JTAG or radio.

- **Sensors**

  Suppressing sensor data by physically destroying or altering a sensor on the hardware board is a very specific attack. It can't be executed very easily, as it requires at least some time and mechanical skills, if the goal is beyond the simple destruction of a mote.

To counter physical attacks, there exist different measures for the different attacks. JTAG ports can be disabled before deployment by intentionally blowing a fuse on the board. The BSL can be set to require a password. However, there exists practically no defense against the soldering or destruction of individual sensors or the whole board.

### 2.4.2 Passive Attacks

In contrast to active attacks, *passive attacks* are executed with the goal to learn about or to make use of information from a system, but they do not affect system resources of that system [92]. They also aim to access restricted information, but without influencing the system's resources. This makes it virtually impossible to detect passive attacks, as no noticeable changes occur in the system.

#### Eavesdropping

The most obvious attack for an attacker is to listen to radio traffic between other nodes. In the simplest case, with unencrypted communication, the adversary might be able to pick up all information that they desire.
The immediate countermeasure is, thus, the encryption of messages. Even if an attacker can pick up encrypted messages, a suitable cryptosystem makes it infeasible to recover the message content without the corresponding key.

**Traffic Analysis**

A step beyond simple eavesdropping is traffic analysis [84]. The IETF defines it as the act of acquiring knowledge of information by inference from observable characteristics of data flow [92]. This includes frequency and size of messages, flow direction as well as the identity of sender and receiver. This could be used, for instance, to identify the data sink in a WSN as target for further attacks.
There exist several proposed solutions to hide the identity of users and to obfuscate the message characteristics. They usually include protocols that try to disguise the routing and, thus, hide the identities of sender and receiver [84, 27].

## 2.5 Security Notation and Fundamentals

In order to accurately describe the security fundamentals in the proposed solution, some notations and definitions are introduced in the following sections. They correspond to the common definitions, as they are used in the literature [69, 26]:

| | |
|---|---|
| $A$ | Identity of Alice |
| $B$ | Identity of Bob |
| $X_A$ | Public key of Alice |
| $x_A$ | Private key of Alice |
| $X_B$ | Public key of Bob |
| $x_B$ | Private key of Bob |
| $Sig_A(M)$ | Signature of message M with public key of Alice |
| $E\{M\}_K$ | Symmetric encryption of message M with key K |
| $D\{C\}_K$ | Symmetric decryption of ciphertext C with key K |
| $K_{AB}$ | Shared secret between Alice and Bob |
| $K$ | Pre-shared secret between two entities |
| $H(M)$ | Hash of message M |
| $MAC(M)_K$ | Message Authentication Code for message M with key K |
| $P(x_p, y_p)$ | Point P with the x-coordinate $x_p$ and y-coordinate $y_p$ |

Further, the message text to be encrypted is referred to as *plaintext*. The encrypted plaintext is called *ciphertext*. The algorithm that is performed for encryption and decryption is the *cipher*. Every cipher uses a *key* for cryptographic operations, that is, a piece of auxiliary information that determines the output of the cipher. In encryption and decryption, the key specifies the particular transformation within the cipher algorithm. Breaking the cipher, i.e. retrieving the contents of an encrypted message without knowing the key, is called *cryptanalysis*. The communicating parties are often referred to as Alice and Bob, with Eve as a malicious attacker.

### Symmetric Cryptography

A cipher that uses the same key for encryption decryption is called *symmetric*, the key is a *symmetric key*, often also called *secret key*. Symmetric encryption provides a secure communication channel between two parties. To set up this channel, two parties, Alice and Bob, agree on a symmetric key that is kept secret from everyone else. When Alice wants to transmit information to Bob, without allowing any other party to access the message contents, she encrypts the plaintext message with the secret key and sends the ciphertext to Bob. He can use the same secret key to decrypt the ciphertext upon receipt to regain the plaintext. Typical examples for symmetric-key ciphers are the Advanced Encryption Standard (AES), Data Encryption Standard (DES) and Rivest Cipher 4 (RC4). In Figure 2.5 the process of sending an encrypted message from Alice to Bob is outlined: Alice and Bob agree on a shared secret (yellow key symbol), for instance by executing the Diffie-Hellman key exchange. Then, the shared key can be used by Alice to encrypt the plaintext and by Bob to decrypt the ciphertext. Dashed arrows denote communications between entities, solid lines show the data flow within a single entity.

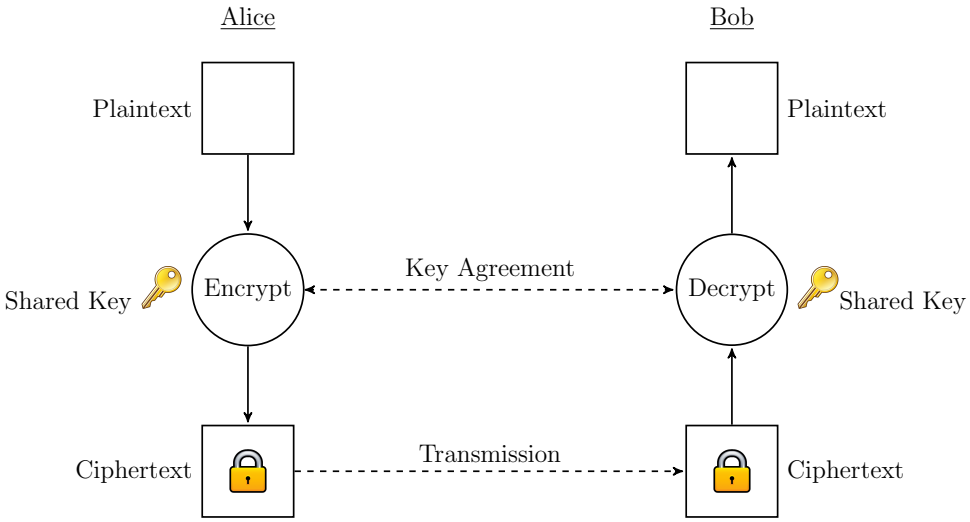



Figure 2.5: Symmetric Encryption illustrated.

One disadvantage of symmetric encryption is the number of keys that each entity has to store in order to enable encryption between all entities in a large network. It is necessary for every pair of communication partners to establish a shared secret. This means for $n$ users, there exist $n(n-1)/2$ keys. Thus, the number of keys increases with $O(n^2)$. On the plus side, symmetric encryption allows for so called *stream ciphers*, that can encrypt an endless stream of plaintext without size restrictions, one bit at a time. By using the the key in a pseudorandom number generator, which creates an endless but deterministic key stream, the symmetric key can be used to encrypt and decrypt data of any size.

### Asymmetric Cryptography

When a cryptographic algorithm requires different keys for encryption and decryption, the cryptographic process is called *asymmetric cryptography*, or also *public-key cryptography* (PKC). The two different keys are labeled according to their respective use: one is kept privately by the owner, that is the *private key*, the other is publicly available to everyone, named the *public key*. The combination of a public key and its corresponding private key is called a *key pair*. Encryption and decryption works both ways, either by using the private key for encryption and the public key for decryption, or vice versa. Applying a cipher that uses asymmetric cryptography is often called *public-key encryption* (PKE). Figure 2.6 illustrates the process of sending an encrypted message from Alice to Bob: First, a key pair is generated by both Alice (blue key symbols) and Bob (yellow key symbols), then the public key of Bob is openly exchanged. Alice encrypts the plaintext with the public key of Bob. The subsequently generated ciphertext from Alice can be decrypted by Bob with his corresponding private key, that was kept a secret. Again, dashed lines denote communications between Alice and Bob, while solid lines stand for message flows within single entities.

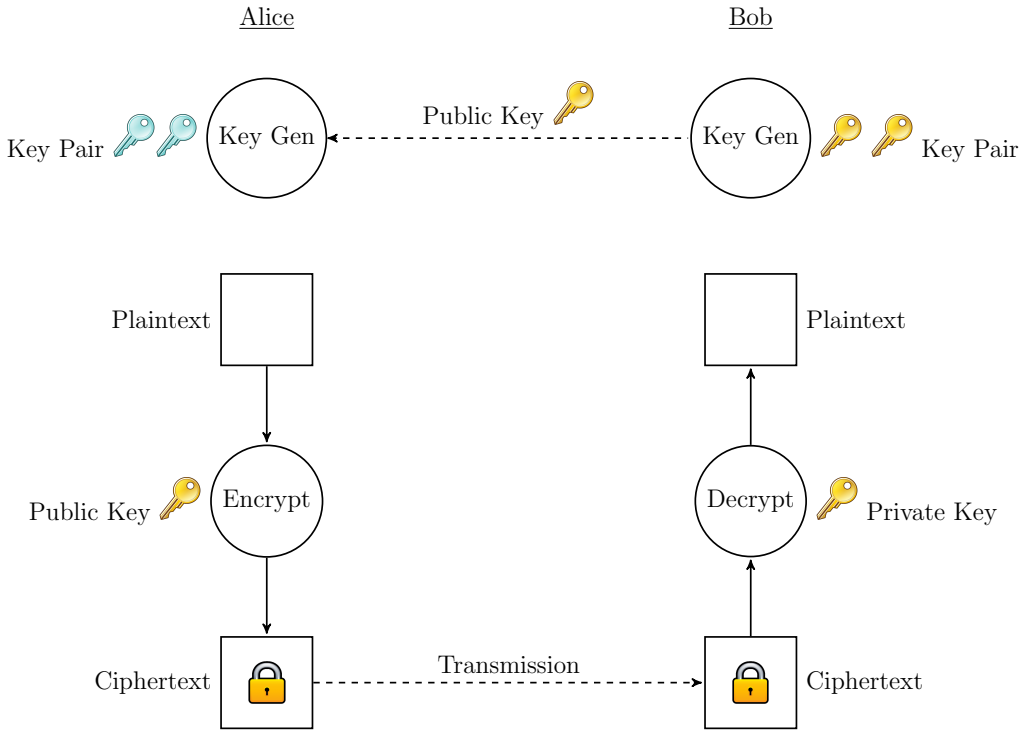



Figure 2.6: Public-key encryption illustrated.

In asymmetric encryption, the number of required keys is much lower than in symmetric encryption. For $n$ users, only $2n$ keys are distributed, since every party can use the same public key to encrypt messages for a particular recipient. Thus, the number of keys only increases with $O(n)$. However, the length of plaintext that can be encrypted with symmetric encryption is somewhat limited: Due to the required mathematical coupling between private key and public key, it is not possible to extend the encryption key in the same way as symmetric keys, as the dependency between keys cannot be preserved. For

example, in the traditional RSA encryption scheme, a 1024 bit key can be used to encrypt at most 117 byte of plaintext at a time (the key is 128 byte, minus 11 byte of padding for security) [26]. Splitting the message into smaller blocks and encrypting each individually is much slower than a symmetric stream cipher.

### Hybrid Cryptosystems

A hybrid scheme combines mechanics from both, public-key cryptosystems and symmetric-key encryption. Public-key encryption as such does not require the computation of a shared secret between two parties, and in symmetric encryption, it is not strictly necessary to store asymmetric keys after the successful computation of the secret. A hybrid cryptosystem, however, uses both asymmetric key pairs and a shared secret for encryption and decryption: A shared secret is computed for every message exchange, the message encrypted with the secret, and the secret itself is encrypted with the public key of the recipient and included in the message as ciphertext. This asymmetric encryption of the secret is called *key encapsulation* [25]. Figure 2.7 shows the process of sending an encrypted message from Alice to Bob: beginning with the key generation by Alice (blue key symbols) and Bob (yellow key symbols), the key exchange of Bob's public key, and the generation of the session key (green key symbol) by Alice. Then Alice encrypts the plaintext with the session key and encapsulation the resulting ciphertext together with the session key in a new message. This is then encrypted with Bob's public key. For decryption, Bob uses his private key for de-encapsulation of the session key and ciphertext and uses the included session key to decrypt the ciphertext back into the original plaintext.
Thus, the hybrid cryptosystem unites the strengths from both symmetric and asymmetric encryption: It requires only $O(n)$ keys, but allows for fast encryption of arbitrarily sized plaintext.

### Signatures

Signatures are used to authenticate messages by including an encrypted hash value of the message. The process of creating this authentication is called *signing*, checking for its correctness is called *verifying* the signature. Depending on the outcome of verification, the signature is either rejected or accepted. Message authentication can increase the trust in message contents, as the recipient is able to check if the content was manipulated (thus includes an invalid hash value) or sent from a malicious attacker that has assumed a fake identity (and uses an invalid key for encryption). However, the application of signatures increases the computational overhead, as it basically introduces an additional step of hashing and encryption or decryption for every message.

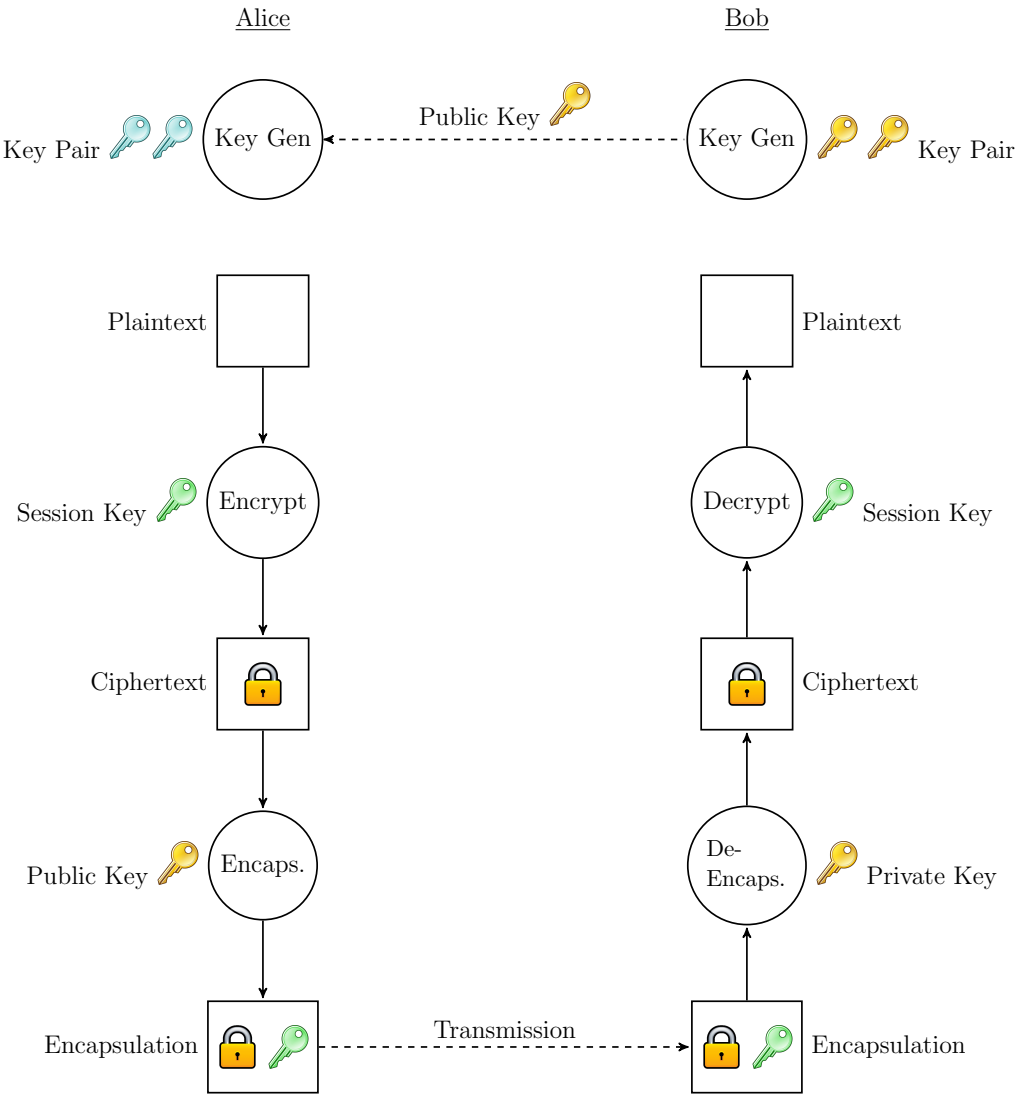



Figure 2.7: Hybrid encryption illustrated.

### Group Theory

To explain the mathematical properties of advanced cryptographic functions, it is necessary to introduce some concepts of group theory first. A *group* is a set $G$ of elements and a binary operation $\circ$ on those elements, written in short as $(G, \circ)$. The operation $\circ$ on elements $a$ and $b$ is denoted as $a \circ b$. In order for a set $G$ to form a group, the following axioms must hold:

- **Closure**: For $a, b \in G$, the operation $a \circ b$ forms another element $c \in G$ .
- **Associativity**: $\forall a, b, c \in G : (a \circ b) \circ c = a \circ (b \circ c)$.
- **Identity element**: $\exists e \in G \mid \forall a \in G : e \circ a = a \circ e = a$, where $e$ is unique and called the *identity element*.
- **Inverse element**: $\forall a \in G : \exists b \in G \mid a \circ b = b \circ a = e$, where e is the identity element.

Further, a group is an *abelian group* if in addition to the above axioms also a fifth axiom holds:

- **Commutativity**: $\forall a, b \in G : a \circ b = b \circ a$.

The group operations on elements are usually addition and multiplication, denoted as $+$ and $\cdot$ respectively. A group is said to be *finite* if only a finite number of elements exists in a group, while all axioms hold for the operation, i.e. the set of elements in $G$ is finite. The finite number of elements in a finite group is the *order* of the group.
A *cyclic* group is defined as a group which is generated by a single element and operation. That is, the group contains an element $g \in G$ such that every element from the group can be generated by repeatedly applying the group operation on $g$. In this case, $g$ is called the *generator* of $G$. Every cyclic group is necessarily also an abelian group.
A group $R$ with a second well-defined operation, i.e. $(R, +, \cdot)$, is denoted as a *ring* if the group is abelian, the second operation (multiplication) is associative and further distributive over the abelian group. A ring, whose nonzero elements form an abelian group under multiplication, is called *field*. In cryptography typically finite fields are used, e.g. a set of integers $\mathbb{Z}$ modulo a prime $p$, written as $\mathbb{F}_p$, which is a finite field of order $p$ (note: $p$ always denotes the order of the field, which in this special case is a result from calculating the modulo with the same $p$). ECC commonly applies two types of finite fields: prime fields $\mathbb{F}_p$, and binary fields $\mathbb{F}_{2^m}$ for $m \in \mathbb{N}_0^+$.

### Discrete Logarithm Problem

Generally, it is required for public-key cryptography to define a function that can compute a matching public key, given a randomly chosen private key. This function should be easy to compute, but difficult to reverse, so that the owner can easily create a key pair, but an attacker cannot reverse-engineer the private key given only the public key. Mathematically, this can be expressed by the equation

$$f(p) = P \tag{2.1}$$

where $P$ is the public key and $p$ the private key. The reverse function

$$f'(P) = p \tag{2.2}$$

should then be computationally hard to calculate. Diffie calls this function $f$ a *trapdoor function* [30]. One often used trapdoor function is the discrete logarithm, like it is applied in RSA and was proposed by Diffie and Hellman in the same paper. It is assumed that in the function

$$b^k = g \tag{2.3}$$

it is easy to compute $g$ when knowing $b$ and $k$, but infeasible to find an integer $k$ when given $b$ and $g$. This is commonly called the *discrete logarithm problem* (DLP), as $k$ is the solution to the discrete logarithm of the expression. Many ciphers are further based on

the use of abelian groups in combination with the DLP. The Diffie-Hellmann key exchange uses the following trapdoor function, where $b$,$k$, and $q$ are integers and $q$ is also prime:

$$b^k \mod q = g \tag{2.4}$$

Intuitively, it is easy to see how an attacker has to compute $k$ times a multiplication and modulo operation in order to find $k$ by brute force, as there exists no efficient approach to solve the DLP over a group. Hence, the attacker has to check the result for each potential number $k$ [109]:

$$b^k \mod q = \underbrace{b \cdot b \cdot b \cdot \ldots \cdot b}_{k \text{ times}} \mod q \tag{2.5}$$

Using the definitions from Section 2.5, it is possible to express the DLP for PKC in terms of group theory: assume a multiplicative, cyclic group $(G, \cdot)$ with generator $g$, order $n$, and a prime number $q$. The private key $x$ is then computed by taking a random integer from the interval $[1, n-1]$ and the public key $y$ as $y = g^x \mod q$. Finding $x$, when only $g$, $n$, and $q$ are given, is the DLP in $G$.

## 2.6 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is an approach in public-key cryptography, where the algebraic structures of elliptic curves (ECs) are used as basis for cryptographic functions. An elliptic curve $E$ is defined by a cubic equation of the form

$$y^2 = x^3 + ax + b \tag{2.6}$$

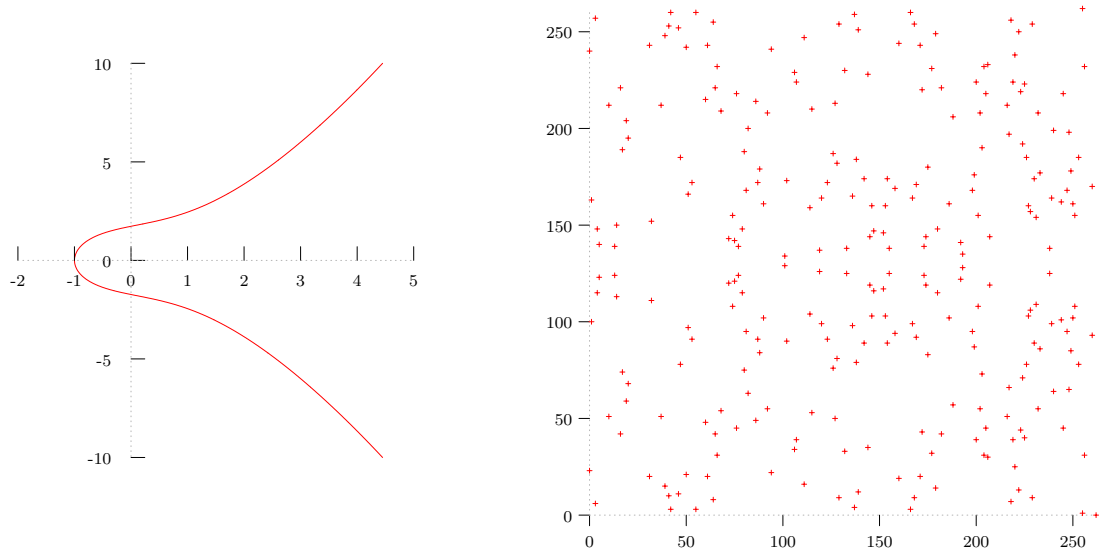
where $a$ and $b$ are constants. The specific curve is written as $E(a, b)$. In ECC, mathematical properties of elliptic curves over finite fields are applied, usually using a prime field [109]. It can be shown that an elliptic curve over a field can be defined in such way that $a, b \in \mathbb{F}_p$ and the following equation holds [109]:

$$4a^3 + 27b^3 \neq 0 \tag{2.7}$$

So the elliptic curve $E$ over the field $\mathbb{F}_p$, written as $E_p(a, b)$, contains the set of all coordinates $(x, y)$ where $x, y \in \mathbb{F}_p$, that satisfy equation (2.6), plus one additional *point of infinity* $O$. In the abelian group that is formed by $E_p(a, b)$, the point $O$ is required as the additive identity element. Figure 2.8a shows a plot of a cure, where $a = 2$ and $b = 3$, i.e. $E(2, 3)$. Figure 2.8b shows the cloud of affine points on $E(2, 3)$ over $\mathbb{F}_{263}$, i.e. $E_{263}(2, 3)$, given by the equation:

$$y^2 = x^3 + 2x + 3 \mod (263) \tag{2.8}$$

According to the definition of finite fields, the set of every point $G$ on the elliptic curve over the field forms a cyclic group. In other words, every point on $E$ over $\mathbb{F}_p$ can be generated by a single element and single operation and is repeated after $n$ operations for




(a) (b)

Figure 2.8: Plot of the elliptic curve $E$: $y^2 = x^3 + 2x + 3$ (a) as plain curve $E(2, 3)$ and (b) over a field as $E_{263}(2, 3)$

order $n$. The generator, here $G$, generates the set $\{0, G, 2 \cdot G, 3 \cdot G, \ldots \quad n \cdot G\}$ of affine points. The order of $G$ is hence the smallest positive integer $i$, so that $i \cdot G = 0$ [109].

The corresponding calculation of $k \cdot G$, for $k \in \mathbb{N}_0^+$, is called *scalar multiplication*, or simply *point multiplication* [109]. Similar to the DLP (cf. Section 2.5), it is assumed to be infeasible to find $k$ when only $k \cdot G$ and $G$ are known and $k$ is sufficiently big. This is called the *Elliptic Curve DLP* (ECDLP) and delivers the trapdoor function on which ECC is based [64]. The ECDLP can be formulated more generally: find $k \in [1, p-1]$ for given EC points $Q$ and $G$, such that

$$Q = k \cdot G \tag{2.9}$$

where $Q$ and $G$ are points on an EC over a finite field $\mathbb{F}_p$. Similar to the DLP in Section 2.5, this is easy to see, since the calculations for a brute force approach need to be executed at least $k$ times [109]:

$$k \cdot G \bmod q = \underbrace{G + G + G + \ldots + G}_{k \text{ times}} \mod q \tag{2.10}$$

Since the arithmetic group operations are applied to EC points (point addition) and EC points with scalars (scalar multiplication), they need to be defined particularly for the group of points on elliptic curves.
The definition of *point addition* can be expressed as follows: a line drawn through two EC points $P(x_p, y_p)$ and $Q(x_q, y_q)$ on the elliptic curve $E$, where $P \neq Q$, will necessarily intersect $E$ in a third point $-R$ (this is a property of elliptic curves). The negative of this point, i.e. $R(x_r, y_r)$ is defined as the result of the point addition of $P$ and $Q$, in other

words $P + Q = R$. This is formally expressed as

$$\lambda = \frac{y_q - y_p}{x_q - x_p} \tag{2.11}$$

$$x_r = \lambda^2 - x_p - x_q \tag{2.12}$$

$$y_r = \lambda(x_p - x_r) - y_r \tag{2.13}$$

Figure 2.9 shows the geometric presentation of this calculation on the curve $y^2 = x^3 - 2x$.

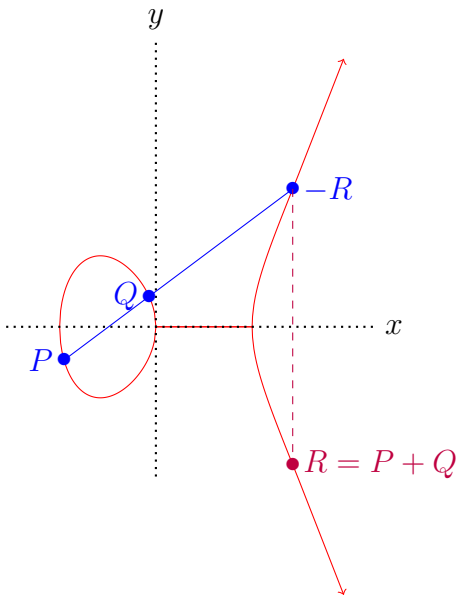



Figure 2.9: Point addition on elliptic curves for $R = P + Q$.

*Point doubling* is similar to point addition, but uses only one point that is added to itself, i.e. $P + P$, which is equivalent to $2 \cdot P$. While in point addition a line between two points was drawn, here the tangent of the point $P(x_p, y_p)$ is used to find the intersection $-R$ with $E$. The negative of this intersection point, $R(x_r, y_r)$, is then the result of point doubling, i.e. $R = 2 \cdot P$. Mathematically, the following equations describe the necessary calculations:

$$\lambda = \frac{3x^2a + a}{2y_p} \tag{2.14}$$

$$x_r = \lambda^2 - 2x_p \tag{2.15}$$

$$y_r = \lambda(x_p - x_r) - y_p \tag{2.16}$$

Figure 2.10 shows the geometric presentation of this calculation, again on the curve $y^2 = x^3 - 2x$.

*Point multiplication* is the repeated execution of point doubling combined with point addition, written as multiplication between a point $P$ and a scalar $k$, where $P \in E$ and $k \in \mathbb{N}_0^+$, for example $R = 5 \cdot P$. There exists no special group operation for point multiplication, instead the factor $k$ is decomposed into multiples of 2, plus 1 for even $k$ or 0 for

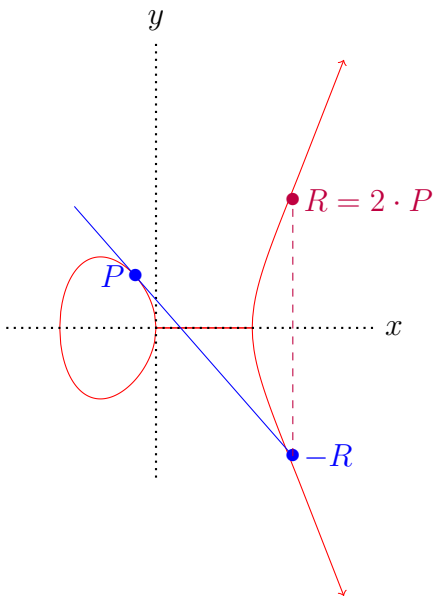



Figure 2.10: Point multiplication on elliptic curves for $R = 2 \cdot P$.

odd $k$. The example $5 \cdot P$ would hence be calculated as $2 \cdot 2 \cdot P + P$, essentially executing point doubling of $P$ twice, then calculating a point addition of the result and $P$.

### 2.6.1 Cryptographic Functions in ECC

As explained in Section 2.6, it is possible to find a trapdoor function, namely the ECDLP, that is based on group operations on elliptic curves. This makes it possible to use the ECDLP for PKC in the same way as it was shown for the DLP in Section 2.5. The general PKC schemes, e.g. for key generation and exchange, need only slight adaption to work with the ECDLP. This section explains the ECC functions that were used in this thesis.

#### Key Generation

Analog to the algorithm shortly described in Section 2.5, an ECC key pair is generated in the following steps [109]:

1. Select curve parameters $a$ and $b$, which define $E(a, b)$.
2. Select a large enough integer q. Either a prime number (for prime fields) or an integer of the form $2^m$ (for binary fields). This gives $E_q(a, b)$.
3. Select a base point, i.e. generator, $G(x_g, y_g)$ on $E_q(a, b)$, with a very large order $n$.
4. Find a random integer $x$, where $x < n$. This integer is the private key.
5. Calculate $X = n \cdot G$. This is the public key and denotes an EC point on the curve $E_q(a, b)$.

### Key Exchange

The ECC key exchange, to generate a shared secret, works similar to the traditional Diffie-Hellman key exchange and is, therefore, called Elliptic Curve Diffie-Hellman (ECDH). It can be executed by performing the following steps [109]:

1. Bob and Alice openly agree on a curve $E_q(a, b)$.
2. Bob generates an ECC key pair, as shown before. This gives him the private key $x_B$ and the public key $X_B$. He transmits $X_B$ to Alice.
3. Alice generates her key pair $X_A$ and $x_A$ in the same way and also transmits $X_A$ to Bob.
4. Alice generates the shared secret key $K_{AB}$ as $x_A \cdot X_B$.
5. Bob generates the same secret $K_{AB}$ as $x_B \cdot X_A$.

Since the resulting shared secret is again an EC point, it is necessary to transform it into an integer for further computations. This is simply done by taking the x-coordinate of $K_{AB}$ as new value. The correctness of $K_{AB}$ can be shown directly by applying the axioms from the group theory in Section 2.5 to the key generation rules:

$$x_A \cdot X_B = x_A \cdot (x_B \cdot G) = x_B \cdot (x_A \cdot G) = x_B \cdot X_A \tag{2.17}$$

### Encryption and Decryption

The encryption and decryption in ECC is generally similar to other approaches in PKC. For symmetric ciphers, the scalar shared secret from an EC Diffie-Hellman key exchange can be used in the same way as any secret from an original Diffie-Hellman key exchange. It can be applied, for instance, to AES, DES, or any other symmetric cipher.
This is more complicated for asymmetric encryption, since the public key is a coordinate on an EC, instead of a single integer and calculations do not hold for just one component of the coordinates. Therefore, a cryptographic system, like ElGamal, cannot be adapted without changes to the algorithm. Bellare and Rogaway have proposed such a variant of the ElGamal encryption scheme, called Elliptic Curve Integrated Encryption Scheme (ECIES) [1]. ECIES is a hybrid cryptosystem (cf. Section 2.5), like ElGamal, but based on the ECDH key exchange. It uses an ECDH generated shared secret between two parties to create two symmetric keys, $k_1$ and $k_2$. Key $k_1$ is then used for symmetric message encryption, $k_2$ to authenticate the ciphertext. The key generation is implemented as a so-called *Key Derivation Function* (KDF), taking a symmetric key as input (the *base key*) and creating one or more new secret keys (the *derived keys*) from the base key [69]. In the case of ECIES, the KDF is constructed from a hash function, taking the base key concatenated with a counter as input. The counter is incremented and the results concatenated, until the required bit length of hash values has been generated [1]. For symmetric encryption, AES is proposed by the International Organization for

Standardization (ISO) [93]. Assuming two parties have already agreed on parameters and exchanged their public keys, the ECIES encryption algorithm for encryption of a message by Alice then works as follows [41]:

1. Select a random integer $r \in [1, n-1]$, where $n$ is the group order.
2. Calculate an one-time key $K = r \cdot G$.
3. Calculate a shared secret $S = r \cdot X_A$.
4. Use a KDF to calculate two derived keys $k_1 \parallel k_2 = KDF(S)$.
5. Encrypt the message with one of the keys, i.e. $C = E\{M\}_{k_1}$.
6. Calculate the authentication tag with the other key, i.e. $t = MAC(M)_{k_2}$.
7. Concatenate $K \parallel C \parallel t$.

The resulting concatenation is the encrypted and authenticated message from Alice, that can be sent to Bob. Upon receipt, Bob has to run the following algorithm to decrypt the message:

1. Split $K \parallel C \parallel t$ into its original elements.
2. Calculate the shared secret $S = x_B \cdot K$.
3. Use the KDF to derive the same two keys as Alice, i.e. $k_1 \parallel k_2 = KDF(S)$.
4. Check if $MAC(M)_{k_2} \equiv t$, reject the message otherwise.
5. Decrypt the message with the second derived key, $M = D\{C\}_{k_1}$.

So in essence, ECIES uses PKC to derive an ephemeral (generated once for each execution of a key establishment process) EC key pair $r, K$. Alice, who knows about $r$ and $X_B$, can compute the ephemeral secret $S$ and tell Bob about the public key $K$. Bob, who thus knows $x_B$ and $K$, can derive the same ephemeral Secret S, since $r \cdot X_B = x_B \cdot K$ (see the description of key exchange above). The secret S is subsequently used for symmetric encryption and authentication, by deriving two symmetric keys via the KDF.

### Signatures

The most used signature algorithm for ECC is based on DSA and, thus, called ECDSA. Similar to DSA, there are two algorithms defined: One for message signing, one for signature verification. The next example assumes that Alice wants to sign a message $M$ to Bob, while both parties have already agreed on parameters, computed their respective ECC key pairs, and exchanged public keys. In order to sign the message, Alice has then to do the following [41]:

1. Calculate $e = H(M)$.
2. Select a random integer $k \in [1, n-1]$, where $n$ is the group order.
3. Calculate the curve point $P(x_p, y_p) = k \cdot G$.
4. Calculate $r = x_p \bmod n$. If $r = 0$, go back to 2.
5. Calculate $s = k^{-1}(e + x_A \cdot r) \bmod n$. If $s = 0$, go back to 2.

The resulting pair $(r, s)$ is then the signature, signed with $x_A$. It is important to select a new and random secret $k$ for every signature, otherwise it is possible to recover the private key $x_A$ from consecutive messages with high probability [41]. Further, the knowledge of one single $k$ can be used to determine $x_A$, so it is vital to keep $k$ secret.

In order to verify the signature $(r, s)$ from Alice, Bob has to run a similar algorithm [41]:

1. Check if $r$ and $s$ are integers in $[1, n-1]$. If not, reject the signature.
2. Calculate $e = H(M)$.
3. Calculate $w = s^{-1} \bmod n$.
4. Calculate $u_1 = e \cdot w \bmod n$ and $u_2 = r \cdot w \bmod n$.
5. Calculate the curve point $Q(x_q, y_q) = u_1 \cdot G + u_2 \cdot X_A$.
6. Calculate $v = x_q \bmod n$.
7. Check if $v \equiv r$. If yes, accept the signature, otherwise reject it.

### 2.6.2 Key Size and Parameters

As mentioned before, the main advantage of ECC over RSA is the smaller key size for the same level of security. The security level takes into account the number of operations that are necessary to solve the DLP and ECDLP with the respectively best mathematical solving algorithm [41]. A security level of $k$ bits means the best algorithm takes $2^k$ steps to compute a solution. Commonly used as reference is the corresponding bit length of a symmetric encryption scheme, which is then simply called security level. In other words, stating that a 160 bit EC scheme has a security level of 80 bits, means it would take the same time to break a 80 bit symmetric-key encryption scheme and a 160 bit EC scheme. Table 2.4 gives a comparison between RSA and EC key sizes for similar security levels, as reported by the National Institute of Standards and Technology (NIST) [73]. The key size here is the bit length of the DLP parameter $k$ in the DLP equation 2.3 and the order of the field $q$ for the ECDLP equation 2.9.

To help with appropriate parameter selections, the NIST has published a set of fifteen recommended fields and corresponding curves [73]. In particular, they describe five prime fields $\mathbb{F}_p$ for primes $p$ of size 192, 224, 256, 384, and 521 bits, where for every $p$ one

| **Security level** | **RSA key size [bit]** | **EC key size [bit]** |
|---|---|---|
| 80 | 1024 | 160 |
| 112 | 2048 | 224 |
| 128 | 3072 | 256 |
| 192 | 7680 | 384 |
| 256 | 15360 | 521 |

Table 2.4: NIST recommended key sizes [73].

specific elliptic curve is defined. Then there are five binary fields $\mathbb{F}_{2^m}$, for $m$ equal to 163, 233, 283, 409, and 571, where for every $m$ one elliptic curve and one Koblitz curve is provided (Koblitz curves are a specific family of elliptic curves over binary fields [97]). Different combinations of fields and curves are commonly referred to by specific names, although different standardization organizations use different names for the same curves. For example, the Standards for Efficient Cryptography Group (SECG) from Certicom calls a certain curve "secp192r1" [98], the American National Standards Institute (ANSI) refers to it as "prime192v1" in the X9.62 standard [45], and the NIST has named it "NIST P-192" [73].

# Chapter 3

# Related Work

The necessity to provide end-to-end security in WSNs is not entirely new. Over the years a few different approaches have emerged and the most relevant examples will be explained in this chapter, while addressing the shortcomings in the existing solutions. The presented approaches served as motivational base and inspiration for the presented solution in this thesis. Table 3.1 shows a summary of the presented projects.

| **Name** | **Authentication** | **Encryption** | **Requirements** | **Class** |
|---|---|---|---|---|
| PKC | Merkle Tree | ECC/RSA | pre-shared keys | 1 |
| UbiSec&Sens | ZCK | public keys / key chaining | none | 1 |
| CDA | not considered | OPE | specific pre-shared keys | 1 |
| HIP DEX | password | public keys / AES | pre-shared passwords | >2 |
| PAuthKey | ECC Certificates | AES | CA | 1 |
| TinySAM | Otway-Rees | AES | none | 1 |
| TinyDTLS | RSA Certificates | AES | CA | 2 |

Table 3.1: Summary of related work.

The work in this thesis is built to support TinyIPFIX, which is not directly related to security, but offers optimizations for data transmissions. It is a protocol for efficient data exchange in WSNs and based on the IP Flow Information Export (IPFIX) protocol [90]. IPFIX was developed as a more resource friendly alternative to CoAP, SOAP (Simple Object Access Protocol), and XML (Extensible Markup Language). It is specified in the IETF standard RFC 7011 [23]. IPFIX is a push-protocol, meaning that nodes can decide when to collect information, allowing them to save energy between transmissions. To reduce packet sizes, IPFIX divides transmission into two parts. First, a so-called IPFIX template is sent, containing meta-information about data fields, sizes and types of future IPFIX record messages. A succeeding IPFIX record contains the actual data values and

includes a reference to the corresponding template, which can be used by the recipient to derive the context for individual values. This differs from other formats, like JSON (JavaScript Object Notation), where messages always contain contextual information, e.g. in key-value pairs.

## 3.1 Pre-Shared Keys

A simple and often quoted solution is the idea of pre-distributing symmetric keys within the WSN before actual deployment [18, 34, 114]. Advocates of this solution usually argue that it requires the least amount of memory with the highest level of security.
Although this is generally true, Du et al. showed that this approach limits the flexibility of the deployment, the connectivity between nodes and the resilience against attackers [31]. They propose a solution that applies public key authentication with less resource demanding symmetric key operations. In their approach, a one-way hash function is used to authenticate public keys. The basic idea is to allow individual motes to verify that a transmitted public key matches the claimed identity, without relying on a trusted third party. For an exhausting mapping between all keys and identities it would be necessary to store a large amount of keys on every mote in the WSN, which is not feasible. Hence, to reduce the memory requirements, they introduce a hash function, mapping identities to the hash value of the corresponding public key. As a result, motes need only to compare hash values and identities, which requires just a fraction of the memory. To further reduce the memory footprint of hash values, a Merkle tree is introduced, where non-leaf nodes are labeled with the hash of the labels of its children [28].

With a focus on data aggregation, Westhoff et al. [107] have developed a different approach: The concealed data aggregation (CDA) scheme. CDA allows for end-to-end security with data aggregation, also one of the main goals in this thesis. They define privacy homomorphisms (PH) that allow to run aggregation functions without prior decryption of messages. Additively homomorphic PHs support additive operations and multiplicatively homomorphic PHs allow for multiplicative operations on the ciphertext. CDA uses order-preserving encryption (OPE) with topology-aware keying. While this solution provides many of the desired properties, it requires that a key is distributed from a pre-defined pool to a subgroup of nodes that form a reverse-multicast routable region in the WSN.

## 3.2 Elliptic Curve Cryptography

A huge research area is the application of elliptic curve cryptography (ECC) to WSN security solutions, in particular for authentication, since ECC can provide strong security with only small amounts of resources [21, 49, 65]. A detailed explanation of ECC can be found in Section 2.5.
Nie et al. for instance describe the *HIP Diet Exchange* (HIP DEX) protocol for hop-by-hop secure connections [76]. The protocol uses a Diffie-Hellman key exchange for public keys and AES encryption for session key exchange. To reduce the computational requirements, the cryptographic primitives are reduced to the bare minimum, for instance by

removing expensive signature algorithms and any form of cryptographic hash functions. To avoid flooding, cryptographic challenges are included in the first messages of the proposed handshake. Identity authentication is achieved by password authentication within the handshake, which requires peers to know about their respective passwords.

To overcome the problem of authentication between sensor nodes, Porambage et al. present the PAuthKey protocol for application-level end-to-end security [82]. It provides pervasive lightweight authentication and keying mechanisms, allowing nodes to establish secure and authenticated communication channels with peers. PAuthKey employs ECC based implicit certificates, using a trusted central certificate authority (CA) to handle authentication security. Thus, it stands in contrast to other authentication approaches, as certificates are generally considered to be too resource demanding for WSNs.

## 3.3 Authentication Without Infrastructure

Some approaches aim for a solution that can provide mutual authentication, while running in a system without additional hardware like CAs or even any previously shared knowledge.

The *Ubiquitous Security and Sensing in the European Homeland* (UbiSec&Sens) project is working towards a whole toolbox of security-aware components. They propose the Zero Common Knowledge (ZCK) protocol for authentication [108], as it can establish well-defined pairwise security associations between entities, even in the absence of a common security infrastructure and pre-shared secrets. ZCK authentication is based on re-recognition between entities, allowing parties to authenticate any other party that they had dealt with in the past. The authors acknowledge that this does not provide complete security, as it would be required for instance for financial transactions, since the first contact between entities cannot be authenticated this way. However, in a scenario without any form of pre-established knowledge or a trusted third party, ZCK provides the best that can be achieved. The ZCK protocol as such does not cater for a key exchange, but can be used in combination with any form of cryptography. The authors present a key-change scheme, public-key and symmetric encryption as examples and devise an improved form of the key-chain scheme that uses hash-chains and largely outperforms many of the alternative solutions [105].

Another solution, providing authentication and security for WSN motes, is TinySAM [66]. This protocol delivers a key management scheme that is based on symmetric cryptography and allows to establish secure and authenticated communications between sensor nodes. The Otway-Rees protocol is applied for one-way authentication, symmetric cryptography with AES is used for encryption. TinySAM requires only minimal resources and no prior knowledge exchange, which is suited perfectly for applications on WSN nodes. It further fully supports data aggregation and IPFIX. The drawback of this solution is the one-way authentication, which leaves the system vulnerable to malicious adversaries, e.g. by allowing the man-in-the-middle attack. Further, it relies on symmetric message encryption with AES, which does not allow for certain other cryptographic mechanisms like message authentication codes or signatures.

A project with a slightly different approach towards hardware requirements is TinyDTLS, where specifically sensor motes with more resources, i.e. class 2 devices, are targeted [57]. One form of class 2 devices are motes with trusted platform modules (TPM). TPMs offer additional dedicated memory and computational power for security functions, allowing for costly computations on otherwise limited hardware platforms. TinyDTLS provides complete Datagram Transport Layer Security (DTLS), which is an adaption of SSL/TLS for unreliable transport layers, such as UDP. TinyDTLS performs a TLS handshake, using RSA certificates for authentication and AES for encryption, but still outperforms most alternatives due to the hight amount of available resources on its target devices. A clear advantage of this solution is the compatibility with established standard protocols, but due to the high demand in resources, devices of class 0 or class 1 are unable to run TinyDTLS.

# Chapter 4

# Design

This chapter contains an in-depth description of the handshake design. It starts with the outline of general security considerations, including an analysis of the threats that were listed in Section 2.4. Those threats are evaluated against the given requirements and assessed accordingly. Following up is a survey of security goals that are ideally to pursuit in a security solution. The last part contains the evolutionary development of the subsequent handshake design, which ultimately is the main contribution of this thesis. As seen in Section 3, there exists no practical solution for class 1 devices until now that provides asymmetric encryption, strong authentication with protection against attacks like man-in-the middle and both without any additional hardware or infrastructure like certificate authorities or trusted platform modules.

## 4.1 Security Considerations

Arguably, an ideal solution should work generically on WSN motes of all classes, especially since the trend goes towards inhomogeneous WSN setups. However, since WSN nodes are primarily designed to collect data, they prioritize cheapness and long liveness over processing power and memory size. As explained in Section 2.1.1, class 0 devices are per definition in RFC 7228 too constrained to run security schemes beyond very specific and limited special-purpose implementations [15], which sets the lower boundary for a target solution to class 1 devices. But even though class 1 devices can connect to the Internet without additional proxies or gateways, they cannot easily communicate with other Internet nodes that employ a full protocol stack (e.g. Transport Layer Security (TLS)). Those devices can be included in IP networks as fully developed peers, but require a specifically designed protocol stack for constrained devices, like CoAP over UDP [15]. Hence, traditional security concepts for wireless networks, such as WEP or TLS, are unsuitable for WSNs [88].
One approach to adapt the traditional Public Key Cryptography to WSNs (cf. Section 3) is the integration of TPMs into motes [31]. By providing a dedicated microprocessor and memory for security, the TPM works independently from main applications and mote processor. Similarly, one could select a platform with more resources on the primary

hardware components in the first place, i.e. class 2 devices. Those devices can deliver Internet-level security, by providing confidentiality and message authentication, at high speed [57]. Hu et al. have shown that a TPM chip outperforms most alternative solutions of similar resource levels [44]. However, and this is a key problem, all motes in a WSN need to be equipped with an appropriate amount of resources to apply such a security scheme network wide.
Hence, in this thesis, the goal is to achieve end-to-end security for devices of class 1 without any additional hardware components or resources. For this reasons, Public Key Infrastructure (PKI) with dedicated certificate authorities is not applicable. It is unrealistic to assume that a WSN of class 1 devices can build and maintain a RFC compliant PKI while still executing its main task, which most likely already takes up most of the available resources: one commonly used OpenSSL X.509 RSA-1024 certificate alone has a size of roughly 800 byte in the IETF specified Privacy Enhanced Mail (PEM) format [63], plus the corresponding RSA key pair takes about another 800 byte [102]. An Aggregator, as defined in Section 2.2, would need to hold $n + 2$ certificates and $n + 2$ key pairs for a degree of aggregation of $n$, quickly filling the available memory. For instance, following those calculations, an Aggregator with $doa = 5$ would need to store additional 11.2 KB, just for certificates and key pairs.
This extreme memory consumption can be avoided by utilizing public key cryptography (cf. Section 2.5). This way, every mote, Aggregator or Collector, has to store only its own key pair and the public key of the current sink. Gura et al. have shown the general feasibility of PKC (cf. Section 2.5) on simple 8-bit processors, like they are typically found on WSN motes. Therefore, in this thesis, a security solution based on public key cryptography is provided. To further reduce the memory footprint and energy consumption, ECC is applied instead of RSA for key generation, key exchange, signatures and encryption (c.f. Section 2.5). ECC requires a smaller key-size, while achieving the same level of security as RSA. The NIST recommendations SP 800-57 explain that a 2048 bit RSA key delivers the same security level as a 160 bit ECC key, i.e. the same amount of resources is required to break them (cf. Section 2.5) [7]. Further, Arvinderpal et al. have shown ECC implementations to be faster and to require less energy, compared to equally secure RSA algorithms [102].
The proposed solution should further run on the application layer, to provide as much portability as possible. Given the modular network stack (cf. Section 2.3), it should be possible to freely exchange individual services, for instance to change the routing protocol from RPL to any other approach.

Of course it would be desirable to address and counter all the potential attacks outlined in Section 2.4. Unfortunately, it is by definition infeasible to implement all countermeasures on highly constrained class 1 devices, as explained in Section 2.1.1. Therefore, the threat levels of different attacks are considered in the scenario together with concepts on how to prevent them. Table 4.1 gives a short overview of the results, while the following section elaborates them in greater detail. In general, the very specific setup of WSNs, especially the predetermined sink and Aggregator nodes, renders some of the attacks less relevant and some others even more severe. Note, however, that the goal of this thesis was specifically to provide a certain set of security functions, which for themselves do not address all the potential issues.

The first attack that was outlined is the **DoS attack**. Such attacks on a single Collector

may not impact the overall WSN. But on an Aggregator, or even the Gateway, it can easily block large amounts of the total traffic, effectively rendering the WSN unusable. This is a high threat and unfortunately yet to be solved. As the proposed solution is built on the protocol and application layer, BLIP is utilized as generic IP stack. Solutions, such as frequency hopping, are, thus, not part of this thesis and would need to be implemented on lower layers. However the spoofing of routing tables is countered by the integration of the RPL protocol (cf. Section 2.3). RPL is an IP based protocol, that uses link-layer quality to calculate routes, therefore preventing the spoofing to a given degree.
The flooding with open connections is prevented in a way that Collectors do not store connection information beyond the single sink address and Aggregators are programmed to hold only a set amount of connections, well within their memory limits.

A **Sybil attack** could be executed easily in the existing TinyIPFIX system, just by taking the identity of one of the other nodes (in this specific case the IPv6 suffix). If this would be for instance an Aggregator or the Gateway, the attack could easily collect most of the traffic. The threat is high, since this would allow an attacker to gain full access to any data stream with just a spoofed identity, no cryptanalysis necessary. The target solution provides direct validation of identities, countering both forms of Sybil attacks.

**Sinkhole attacks** pose a medium threat to the target setup, as they can be used to interrupt message flows. However, they are not easy to execute: routing relies on the RPL protocol in the current implementation (see DoS above), that uses link-layer quality to calculate routes, which is already one of the proposed counter-measures against sinkhole attacks [53, 101].

**Hello flood attacks** are again very relevant to the use-case and pose a high threat level: Aggregators announce their presence with a simple hello message, containing only their identity. The identity can easily be forged (cf. Sybil attack). However, the intended solution provides direct identity validation, avoiding the forging of identities and consequently also a successful execution of hello flooding.

**Man-in-the-middle attacks** are highly relevant to the given scenario and one of the main issues with existing solutions. A successful MITM attack allows an adversary to intercept and alter any data stream, even if it is encrypted, without detection. The proposed two-way authentication protocol solves this issue, by providing authentication between peers that an attacker cannot forge.

The relevancy of **physical attacks** depends highly on the target application. In an experimental setup, nodes are theoretically completely open at all three physical attack points (JTAG, BSL, and sensor hardware). Nonetheless, one must argue that this is a very specific threat that requires physical access to motes and is not really relevant to the research within the scope of this thesis. Hence it should not be seen as a threat on the application level. Countering any attacks, e.g. by blowing a fuse to disable a JTAG connector, is counterproductive in a research environment. It is in the responsibility of a potential enduser to address those threats upon deployment.

**Eavesdropping** is an attack against one of the main goals of this thesis. It is intended to provide secure end-to-end communication, which does not allow for unauthorized access

to transmitted data. This thread is consequently categorized as high and countered by strong end-to-end encryption.

**Traffic Analysis** is a lower threat. While it is often the starting point for other attack types, it alone is neither harmful, nor does it expose sensitive message contents as such. This work does not address this weakness explicitly, as counter measures would (similarly to the DoS attacks) require changes in the lower layers. It is acknowledged as an open issue, but implies no immediate need for action.

As summary, Table 4.1 illustrates the cumulative points from above. The notation is as follows: "X" means that protection is possible in the proposed solution. "O" stands for open issues that were not considered due to low threats. "+" denotes an attack that is not solvable on the application layer, but needs to be addressed on other layers. "-" means that those attacks were not considered at all, since no protection is possible with software implementations alone.

| **Attack form** | **Attack** | | **Threat level** | **Solution** |
|---|---|---|---|---|
| Active | DoS | Physical Layer | high | + |
| | | Data Link Layer | high | + |
| | | Network Layer | high | X |
| | | Transport Layer | high | X |
| | Sybil | Fabricated Identity | high | X |
| | | Stolen Identity | high | X |
| | Sinkhole | | medium | X |
| | Hello Flood | | high | X |
| | Man-in-the-middle | | high | X |
| | Physical | JTAG | low | - |
| | | BSL | low | - |
| | | Sensors | low | - |
| Passive | Eavesdropping | | high | X |
| | Traffic Analysis | | low | O |

Table 4.1: Threat levels and protection in the proposed implementation.

## 4.2 Security Goals

In this section, the goals for the security solution are defined. As explained in previous sections, the objective is to provide a solution for very constrained devices of class 1 or above. Hence, the first step is to summarize from the previously made observations what goals *should* be implemented and later in Section 6.4 compare them with what actually *can* be implemented on the limited resources.
In general, institutions like the NIST, the IETF and other researchers mostly agree on a set of security objectives that are necessary to achieve information security. Those objectives are confidentiality, integrity, authenticity, availability, and accountability of all messages [111, 53, 99]:

- **Confidentiality**

  Private or confidential data must not be disclosed to unauthorized individuals. This applies to data in storage, during processing, and while in transit. In the case of WSNs, this equals the requirement that attackers cannot gain knowledge about the content of a secured message and from physically accessible motes.

- **Integrity**

  Integrity ensures that a message, which was transmitted from one node to another, is not modified by a malicious attacker. If modifications have occurred, they must be detected. In WSNs, this applies to all radio-transmitted messages.

- **Authenticity**

  Communication partners can be reliably identified and any changes to sender information can be detected. In other words, receivers must be sure that a sender is who he claims to be. Again all radio-transmitted messages are relevant to this objective in WSNs.

- **Availability**

  A system should always be available to legitimate users and deliver prompt and reliable service. This is especially relevant in the case of ongoing DoS attacks. In WSNs, the availability of individual motes and the Gateway are in the focus of this objective.

- **Accountability**

  All actions - authorized or unauthorized - should be uniquely traceable to the responsible entity. Especially the active integration of Aggregator nodes in WSNs is relevant for this topic.

Furthermore, a set of requirements that are particular to WSNs and the development goals for this thesis must be considered:

- **End-to-End Security**

  Wireless networks are generally easy to overhear and easy to spoof, since anyone in range can access the medium. Thus, an end-to-end protocol provides security, even if the underlying network infrastructure is only partially under the user's control and might be compromised. Especially in a WSN, where multi-hop routing is used during authentication and key exchange, this is a very important design goal.

- **Support for Unreliable Transport Protocols**

  While TCP performs well in traditional networks, it suffers from a severe performance drop in wireless networks, due to lossy connections that invoke congestion control algorithms [6]. Hence, the proposed solutions uses UDP for data transmissions.

- **Two-way Authentication**

  Also called *mutual authentication*. It denotes two entities authenticating each other at the same time [69]. In the scope of WSNs, it is not enough to authenticate only the sender to the receiver, but the sender should also be sure about the identity and authorization of the potential receiver of sensitive data.

- **Elliptic Curve Cryptography**

  To save memory, energy and computational power, one main goal of this thesis is to utilize ECC for all cryptographic purposes. For message encryption, public key cryptography is preferred over symmetric encryption algorithms.

## 4.3 Protocol Design

The key contribution of this thesis is the design of a two-way authentication handshake protocol for class 1 devices. The final design is a direct result of the aforementioned security considerations (cf. Section 4.1), while keeping in mind the extremely limited resources of sensor motes. This section introduces definitions, notations and outlines the evolutionary steps of the handshake specification with respect to the security goals (cf. Section 4.2).

### 4.3.1 Definitions for Protocols

A few definitions are necessary to sufficiently describe some of the properties of key exchange protocols: The particular instantiation of a protocol is called *run* [29]. The run is said to be successful if both parties do not only exchange keys, but also accept them as valid. For this, each participant has to confirm that the key exchange was successful for both parties. This leads to the authentication of the shared secret. *Explicit key authentication* assures that a) no other party aside from the intended partner could have knowledge of the shared key and b) the counterpart definitely is in possession of the key. If only a), and not b) is guaranteed, one usually speaks of *implicit key authentication* [39].

### 4.3.2 Protocol Properties

Diffie et al. argue that an authentication protocol should always be linked to the key exchange for later encryption, otherwise an attacker might just wait until authentication is completed and compromise the established communication channel [29].
Canetti et al. then summarize the objective of a key exchange protocol in a very intuitive way: According to them a key exchange protocol is *secure* if it is impossible, or at least infeasible, for an attacker to distinguish the generated key from a random value [19]. The same fundamental concept can be applied to an *authenticated key exchange protocol* (AKE). But additionally, *entity authentication* should guarantee the identity of the

communicating parties in the current communication session, therefore preventing impersonation [26]. A good authentication protocol, thus, combines several properties, as explained by various researchers [10, 12, 60, 11, 29]. The following paragraphs give an outline of those properties, as they are consequently also a basis for the protocol design in this thesis.

#### Forward secrecy

In the case that a generated private key of one or more of the participating entities is compromised, the security of previous communications must not be affected. In the case of asymmetric encryption it can be distinguish between *half* forward secrecy, where only one single entity's key may be compromised, and *full* forward secrecy where keys of both entities can be compromised.

#### Asymmetry

To avoid reflection attacks, where one entity simply replays the same message back to the sender, it is desirable to avoid symmetries. In other words, the authentication responses of two different parties must not be identical.

#### Direct authentication

A protocol provides *direct* authentication if authentication is complete with the end of a protocol run, i.e. both parties have proven knowledge of the shared secret. A protocol with *indirect* authentication requires the transmission of further messages, for instance by encrypting consequent messages with the shared key.

#### No timestamps

To utilize timestamps, every participating entity is required to maintain a reliable local clock, all of which must be synchronized periodically. Especially in WSNs with constrained motes it is not possible to guarantee such a property, as many motes reset the local clock on every restart or power outage (i.e. removal of batteries). Hence, network wide time synchronization would be necessary, which introduces additional complexity and resource consumption [95]. And even in more powerful environments can the utilization of timestamps pose a serious thread, as synchronization failure renders an otherwise secure protocol vulnerable to attacks [40]. For more information on this topic, refer to Gaarder et al., who have made in-depth observations on timestamps in security applications [37].

### 4.3.3 Handshake Evolution

As a starting point for the protocol design, the focus was on a two-way authentication of two independent entities without prior information exchange. As explained in Section 4.3.2, the authentication protocol should always include a key exchange. Hence, the simplest *Station-To-Station* (STS) protocol is an ideal candidate for this first design step. It is based on a Diffie-Hellman key exchange, followed by an exchange of authentication signatures, as outlined by Delfs et al. and based on works of Diffie [26]. Both parties each compute a private and a public key first. Then one party, say Alice, sends her public key to the other party, say Bob. Once Bob receives Alice's public key, he can compute a shared secret according to the Diffie-Hellman key exchange algorithm [69]. Using this *ephemeral* shared secret key, Bob can now encrypt any message to Alice. For her to be able to decrypt the ciphertext, Bob sends his public key back to Alice, so that she can compute the same shared secret.
In the second part, the authentication, Bob sends a token consisting of both public keys, signed with Bob's own private key. Alice can use Bob's public key to verify that it was indeed the same person who signed the message and computed the shared secret. Bob is now authenticated to Alice. A similar authentication message is then built and sent from Alice to authenticate her to Bob. To avoid communication overhead, it is feasible to combine the second key exchange message with the first authentication message. Figure 4.1 shows the subsequent flow of messages between entities.

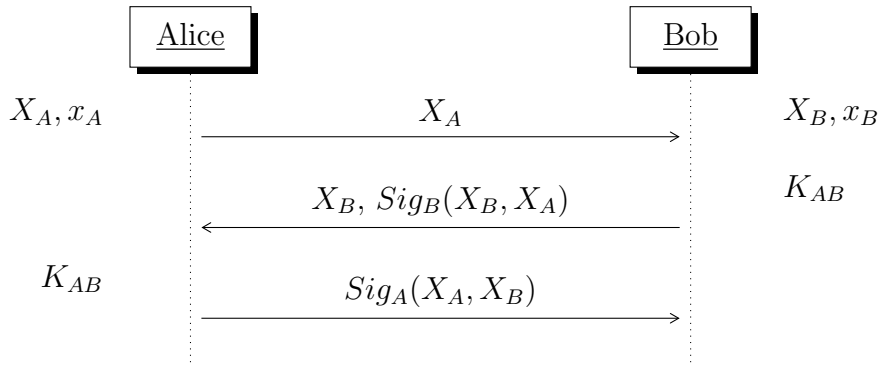



Figure 4.1: Station-To-Station protocol for key exchange and mutual authentication.

The protocol entails the establishment of a shared secret key between two parties with mutual entity authentication and mutual implicit key authentication [69]. Because the shared key is an ephemeral Diffie-Hellman key, the STS protocol provides forward secrecy [16]. The signatures are used to obtain protection against impersonation during the exchange.
However, there are several shortcomings in this STS protocol. While it is relatively easy to execute, it does not include explicit key confirmation: neither Bob, nor Alice can be inherently sure that the other party has actually computed a shared secret without additional messages. Further, the STS protocol is vulnerable to two well-known attacks: Unknown Key-Share Attacks (UKSA) and Man-in-the-Middle (MITM) Attack, which are explained in the following.

**Unknown Key-Share Attacks**
In this attack, an adversary Eve could replace the signature in the last message with her own, making Bob believe that the key is shared with Eve, while Alice believes the key is shared with Bob [13]. Figure 4.2 shows an example for such an attack against Bob. The self-claimed identities of senders are explicitly stated here, to show how Eve spoofs hers, even though they are usually not part of message payloads. Note that in this scenario Eve never gets access to message contents. The only impact is the illusion of Bob that Eve has been correctly authenticated. Blake-Wilson et al. describe a banking scenario as possible example for a successful attack, where Eve can now receive a deposit on an account mapped to her name, even though the deposit request was authenticate by Alice [13].

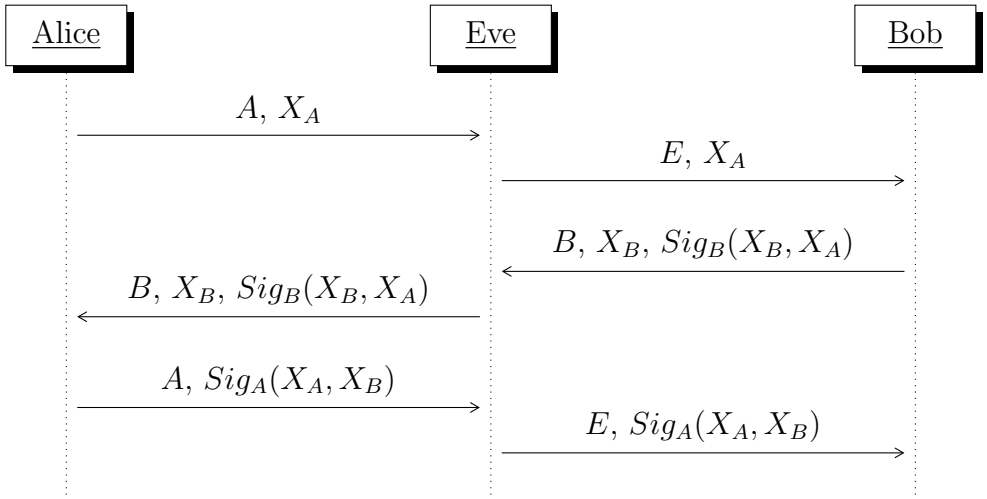



Figure 4.2: Unknown key-share attack on the Station-To-Station protocol.

**Man-in-the-Middle Attack**
Eve, the adversary, puts herself simply in between Alice and Bob. She can then intercept messages in both directions, reading, signing and encrypting them as her own. While Alice and Bob each assume that they communicate with the other, they in fact only communicate with Eve, who forwards the message contents. Figure 4.3 shows the execution of this attack on the STS protocol. Alice wants to establish a connection with Bob, but effectively communicates with Eve. At the same time, Bob assumes that he receives secret messages from Alice. Even though the contents of those messages are indeed from Alice, Eve has full access to them.

Considering for the moment only one of those two vulnerabilities, the basic STS protocol can easily be extended to counter unknown key-share attacks and to provide explicit key authentication. It is only necessary for Alice and Bob to additionally encrypt their signatures with the shared secret $K_{AB}$, provided that both have successfully computed $K_{AB}$. Figure 4.4 shows the resulting protocol design.
Examining the consequential outcome from this adaption, like Diffie et al. propose [29], yields the following: Bob can be assured that he shares $K_{AB}$ with only one single party. Since he has created $X_B$ specifically for this run and Alice has signed $X_B$ and $X_A$, her

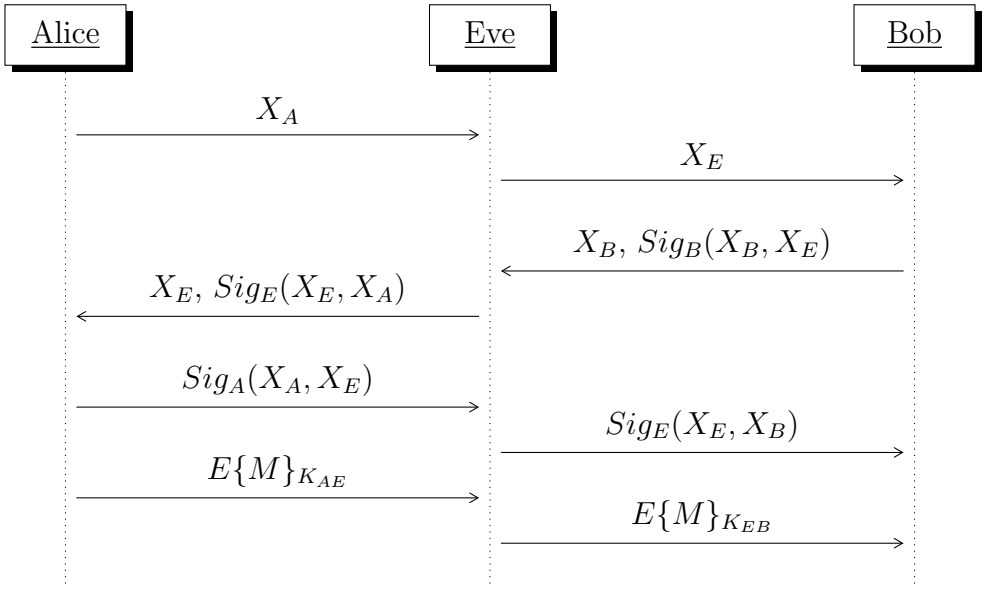



Figure 4.3: Man-in-the-middle attack on the STS protocol.

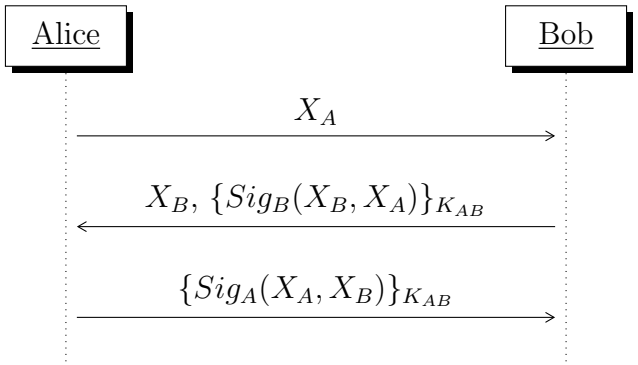



Figure 4.4: Adapted STS protocol for key exchange and mutual authentication.

signature is now tied to this particular run. By encrypting the message with the resulting $K_{AB}$, she assures Bob that she was indeed the entity who created $X_A$. Similar assumptions can be made from the position of Alice.
On the disadvantageous side, it is computationally more expensive to run signature and encryption algorithms together. Furthermore, as the encryption would necessarily be symmetric, one would need to introduce a symmetric encryption algorithm in the system, specifically for this purpose. In the light of very limited memory on motes, it is desirable to avoid this sort of overhead. As Boyd et al. observe, the necessity for encryption can be solved by including the identity of both entities in the exchanged signatures. The resulting protocol, the adapted STS protocol plus identities in signatures, is almost identical to the Bellare-Canetti-Krawczyk, (BCK) protocol, as observed by Blake-Wilson et al. [12, 10, 16]. The only difference in BCK is the absence of the sending entity's identity. According to Basin et al, it is generally desirable to include identities of both parties, but in a bidirectional exchange, as it is the case here, it is strictly speaking only required to include the receiver's identity [8]. Figure 4.5 shows the complete BCK protocol. By including the identities, it becomes impossible to relay signatures under a spoofed identity. At least in one direction, the receiving entity would be presented with a invalid signa-

ture, that does not contain their own identity, and immediately abort the handshake. At this point, the protocol is computationally relatively cheap, but still vulnerable against the second attack form, the man-in-the-middle attack [16]. The weakness against MITM attacks boils down to the fact that it is not possible to reliably map a public key to a specif entity, i.e. to derive their public key from their identity. Any entity can claim any public key as theirs. To counter this, one needs to strongly couple a public key with the respective identity.

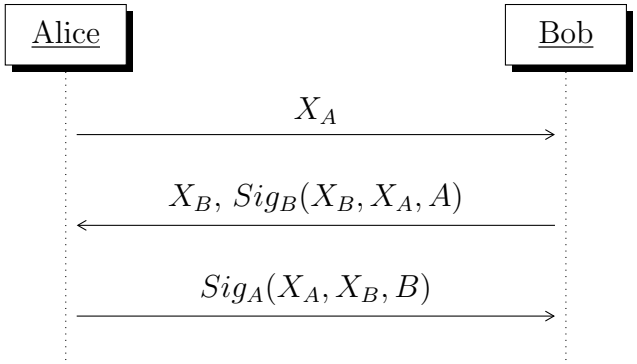



Figure 4.5: BCK protocol, resistant to unknown key-share attacks.

The prevalent solution for this is to introduce public key infrastructure with certificates and trusted certificate authorities, like for instance proposed in the IETF specifications for TLS [14]. A certificate contains the identity and the corresponding public key. As the certificate is constructed by a trusted CA, entities can be assured of the correct coupling between key and identity.
But since one of the goals of this thesis is to avoid any additional infrastructure, this solution does not suit the given requirements. Hence, it is necessary to verify the identity of an entity in another manner. As Badra et al. line out, a pre-shared key (PSK) is also suited to provide authentication [5]. The distribution of pre-shared keys is fairly simple in the context of WSN motes: adding a unique PSK to the programming procedure introduces practically no overhead, since motes need to be programmed before deployment in any case and the key generation and management can be moved to the software that is used to program the motes. Note that this procedure differs from the presented approaches in Section 3, as not every mote is equipped with a set of keys for encryption between peers, but only one PSK for authentication towards the gateway.
Given the assumption that only the two concerned parties have knowledge of the PSK, each party can absolutely be assured that anyone who uses the PSK is indeed the other party. For this to work more than once, it is vital to keep the PSK a secret between the two parties, i.e. not to send it as plaintext during authentication. Otherwise any attacker, who might pick up the message, can use the PSK for herself. In fact, it must be avoided to send any form of information that can a) be used to retrieve the PSK or b) replayed to achieve authentication for any other entity. Traditionally, those two goals are met by transmitting a cryptographic hash digest of the PSK together with a cryptographic nonce. Including a different nonce in every message makes it impossible to use an authentication message a second time, i.e. in a replay-attack. This is called *digest access authentication* and outlined in the IETF specifications RFC 2617 [36]. The goal here is slightly

different, as it is desired to couple a unique public key with the PSK (and consequently the corresponding identity), which may be replayed several times, but never for another public key. Hence, it is possible to use the public key instead of a random nonce and to create a hash from the PSK and this public key, i.e. $H(K, X_A)$. This ensures Bob that $X_A$ is indeed Alice's public key [83, 85]. A cryptographic hash function is infeasible to revert, even with a partially known input (here the public key is publicly known), the PSK is impossible to recover. At the same time, a spoofed hash digest for a different public key can only be produced with the knowledge of the PSK. Consequently, replaying messages would only serve to authenticate an attacker to a node with a public key for which she does not know the private key counterpart. To provide mutual authentication in the authentication protocol, those digests must be computed from both parties, with their respective public key. For reasons of avoiding transmission overhead, the digests can be included in the first and second BCK protocol messages instead of additional messages. The complete handshake is shown in Figure 4.6.

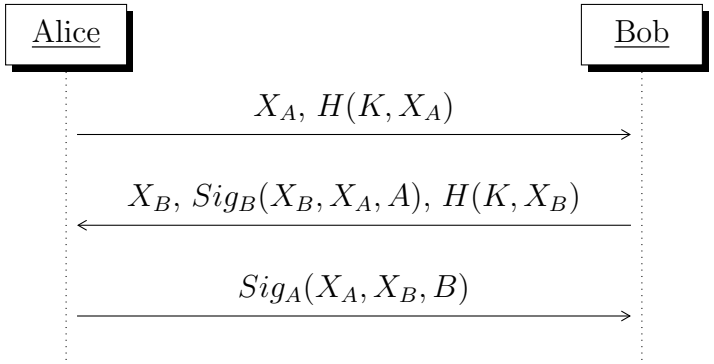



Figure 4.6: Extended BCK protocol with PSK.

In accordance to the previously declared requirements, this is a two-way authentication protocol with included key agreement, delivering direct and explicit key authentication. Messages do not include timestamps, are completely asymmetrical, and cannot be used for replay or reflection attacks. Only forward secrecy cannot be guaranteed by the handshake alone and needs to be addressed by appropriate encryption techniques of the subsequent messages.

### 4.3.4 Aggregator Support

As explained in Section 2.2, WSN often contain several motes that are programmed for different roles. The two flexible roles in this scenario being Collectors and Aggregators, the Gateway is unique and static. Even though both roles can use the specified handshake to establish a secure communication channel with the gateway, Aggregators introduce additional complexity in the system. Collectors need to switch the destination of their data stream from the Gateway to an Aggregator, which in turn needs to processes the information. The simplest, albeit naive, way of doing this would be for the Aggregator to announce its presence to Collectors, which in turn would redirect their stream upon receipt. Figure 4.7 shows the four conceptual steps that are necessary to introduce an

Aggregator into a WSN with two Collectors in this naive way, i.e. without authentication between Aggregator and Collector. This general concept is based on [33], although this approach does apply neither authentication, nor encryption. Introducing the security components leads to the following variation:

1. The starting point is an already deployed WSN, containing two Collectors. Each Collector has completed the handshake with the Gateway and is transmitting data over an encrypted channel. Additionally, an Aggregator is introduced, but not activated yet.

2. The Aggregator is activated, immediately contacts the predefined data sink, and executes the handshake.

3. The Aggregator has established a secure channel with the Gateway. Further, the Aggregator announces its presence to ever Collector in range by broadcasting a short message. This broadcast can take any form, for instance a simple echo request, a counter, or a nonce, as long as Collectors are programmed to wait specifically for this type of message. It this case, one would include the Aggregator's public key, to avoid the additional communication overhead of Collectors requesting the key from the Aggregator.

4. Collectors react by redirecting their streams to the Aggregator, now encrypted with the public key of the Aggregator. The Aggregator decrypts both streams, processes the information, encrypts the results again and sends the new message securely to the Gateway.

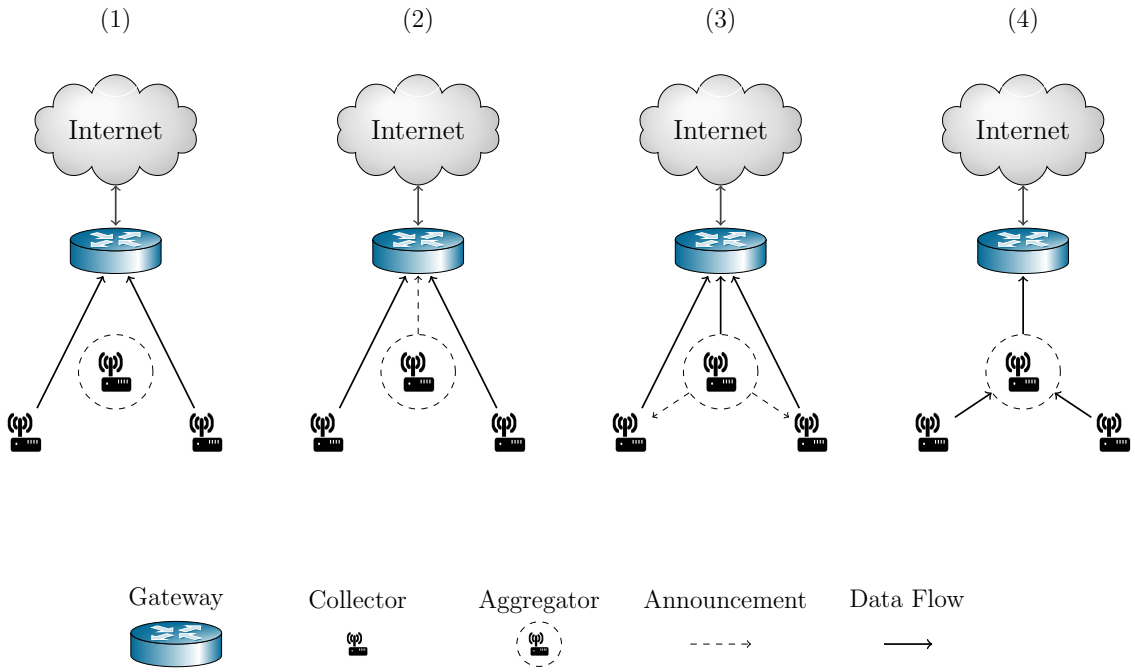



Figure 4.7: General approach to introducing an Aggregator to the WSN.

While this approach provides encryption of messages between all parties and, therefore, protection against eavesdropping between Collectors and Aggregator as well as Aggregator

and Gateway, it entails one important drawback. Collectors have executed the complete handshake with the Gateway, so both parties are mutually authenticated; meaning Collectors are assured about the identity of the Gateway and vice versa. However, in step (3) and (4), Collectors sacrifice all assertions about identities if they blindly react to announcements of Aggregators. An attacker would just need to broadcast an Aggregator announcement to get easy access to data streams from every collector in range, conveniently encrypted with his own public key. Since this would break the whole security concept, it is imperative for Collectors to establish a new secure communication channel to Aggregators, that holds all the previously explained security and design principles without blindly trusting Aggregator broadcasts. Consequently, the authentication needs to be extended to include authorization. A Collector needs confirmation that any broadcasting Aggregator is a valid aggregation node and not an intruder who tries to access sensitive data.
In the basic scenario, where Collectors or Aggregators communicate only with the Gateway, this is implicitly covered by the exchange of pre-programmed PSKs. Intuitively, it is possible to pre-program Aggregators and Collectors with pairwise PSKs in the same way. They would then need to execute the complete handshake, including authentication and key exchange. Although this would work in theory, it subsequently sacrifices any form of flexibility. Aggregators could only aggregate data streams from predefined Collectors and would further need to hold $n + 1$ PSKs for $n$ Collectors.
A more flexible and less resource demanding solution is to lever the already fully authenticated and secured channels between both Aggregator and Collector, and the Gateway. Upon receipt of an Aggregator announcement, the Collectors only needs to check with the fully authenticated Gateway if the broadcast sender is an authorized Aggregator. If it is, the Gateway can reply with the public key of the Aggregator, signed by his own trusted private key. This is similar to a PKI, where the Gateway takes the role of the certificate authority as trusted third party. Figure 4.8 shows the resulting message flow between Aggregator, Collector, and Gateway.

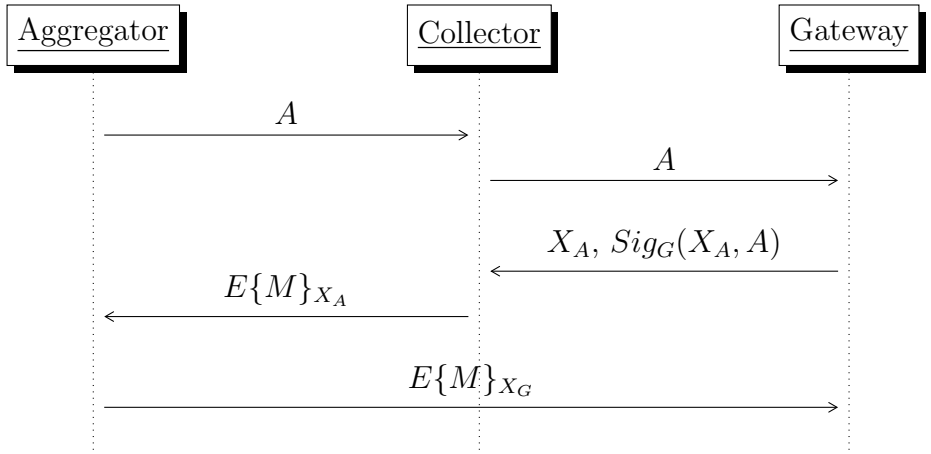



Figure 4.8: Message flow to securely introduce an Aggregator to the WSN.

It is assumed that all parties have previously completed successful runs with the Gateway. The signature is used to verify the mapping between the Aggregator's public key and its identity, making it impossible for attackers to spoof the transmitted public key. This

stands in contrast to the previously exchanges authentication messages, where the identity of the receiving party had to be included instead of the key owner's identity: This is not necessary here, since the channel between Gateway and Collector is already authenticated. The Aggregator's Identity is not encrypted between Collector and Gateway, which can allow attackers to spoof the transmitted identity, since this additional encryption would require expensive computations. Hence, it is substituted with the identity in the signature from the next message, which is computed by the more powerful Gateway. The Gateway might reply with the public key for a spoofed identity, but the Collector can detect this spoof easily due to the invalid signature and abort the process.

# Chapter 5

# Implementation

This chapter describes how the security solution was actually implemented for this thesis. Thus, the basics concepts of TinyOS programming are outlined, followed by the software architecture of the program that is to be deployed on the motes. The resulting implementation includes the full handshake with key exchange and encryption between motes and the gateway. Hence, the solution has to be included not only in the original TinyOS program on motes, but as well as in the existing WSN framework CoMaDa on the Gateway. Further, it has to support both data collection and data aggregation.

## 5.1 TinyOS

The free and open source operating system (OS) TinyOS is used to program WSN motes, currently available in version 2.1.2 [61]. TinyOS is designed specifically to run on sensor networks, with huge limitations on memory and computational power in mind. The core OS is roughly 400 byte, which leaves enough space for complex programs on typical sensor motes (cf. Section 2.1). TinyOS applications are written in nesC, which is an extension of the C language [38]. The program code is statically linked and compiled with a custom GNU tool-chain, based on the make tool.

TinyOS utilizes a component based programming model, allowing the user to build an application-specific OS for every application [61]. The resulting program is a graph of those components, each of which represents an individual software entity. Generally, components expose one or more interfaces and are connected (say *wired*) over those interfaces to form an executable. Some components expose hardware abstractions, which are often provided by the TinyOS core, for instance for routing, sensing, and storage. Hence, every component has to be defined in two scopes, the specification of its interfaces and further the actual implementation. This allows a programmer to easily exchange components: For instance a component for routing that provides the required interface can easily be exchanged for another component that provides the same interface, but uses a different implementation. For this reason, components do not only *provide* interfaces, but also *use* interfaces in their implementation. While the provided interfaces represent the

functionality that a component can offer, the used interfaces specify the functionality of other components, which the defining components needs in order to perform its job.

Due to its minimal design, TinyOS has only one stack and executes function calls (i.e. synchronous *commands*) non-preemptively and non-blocking. To support complex computations over more than a few milliseconds, components may post a *task*, telling the scheduler to run the execution later. Execution is then scheduled after the first in first out principle (FIFO). Tasks run asynchronously, using callbacks to signal successful execution. This offers a rudimentary form of concurrency, but brings its own problems. For example, expensive tasks that run on over several seconds can completely block other function calls, such as queuing received packets, which are then lost.

## 5.2 Software Architecture

The TinyOS core of version 2.1.2 includes hardware abstractions for many different hardware platforms (e.g. Telos family, MicaZ, IRIS) each of which represents a different combination of processor (e.g. MSP430 family, Atmel's Atmega128), radio (e.g. Infineon TDA5250, Atmel RF212 and RF230), and storage (Atmel AT45DB and STMicroelectronics STM25P). Those hardware abstractions are wired together via the corresponding components and are supplemented with components from higher layers. One such high level component is the BLIP IP Stack (c.f. Section 2.3), which is used in version 2.0. It is responsible for the whole network layer in the network stack (cf. Figure 2.4) and handles IPv6 communication in TinyOS programs. This includes 6LoWPAN for packet transmission and the RPL protocol for routing. Figure 5.1 shows the details of the 6LoWPAN/RPL software stack in TinyOS 2.x [55].

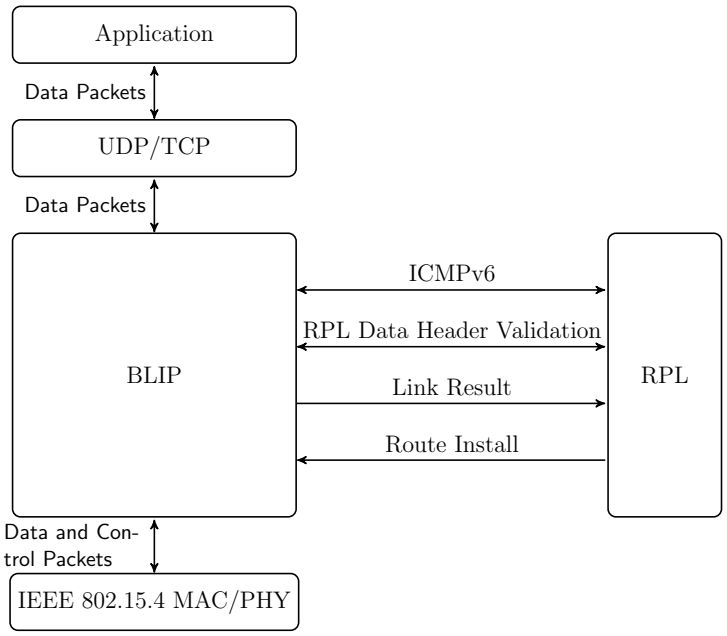



Figure 5.1: The 6LoWPAN/RPL software stack in TinyOS 2.x as seen in [55]

In combination with the included UDP/TCP abstractions, this allows programmers to call TCP and UDP transmission on the highest abstraction level from the application layer. For example, sending an UDP packet from the application layer is reduced to one single line:

```
call UDP.sendto(&sinkAddress, (void*) message, sizeof(message));
```

Thus, it is easy to implement additional functionality on the application layer. A security solution on this layer can be easily encapsulated in a security component, which is wired in between the actual application and the transport layer. Figure 5.2 shows the resulting data flow within a entity (solid lines) and between entities (dashed lines). The TinyTO security component encrypts application data, which is in turn packed into IPv6 payloads, that are handled by BLIP and lower components.

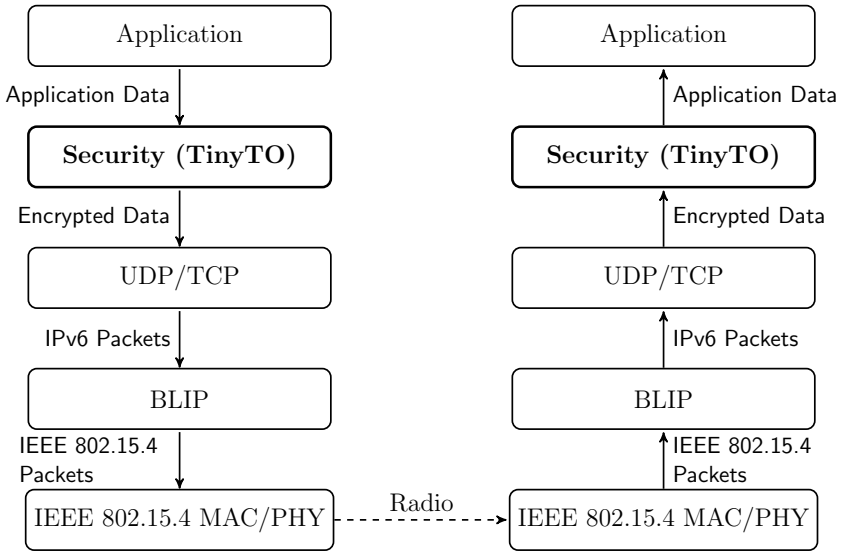



Figure 5.2: Data flow between two entities with security components.

The concrete implementation is based on the existing TinyIPFIX solution [90], which includes one TinyOS application for data collection and one application for data aggregation, both with IPFIX support. As field-tested security solution, the TinyECC library is used, which provides an efficient implementation of ECDH, ECDSA and ECIES [64]. The ECC curve is adaptable, but defaulted to the well-known and NIST recommended secp192k1 curve, thus generating 48 byte sized keys. Communications on transport layer are implemented over UDP, in accordance with the design goals for unreliable communications.

One immediate problem of this approach is the high memory requirement of the combined components. While both TinyIPFIX and TinyECC fit very well on motes individually, the combination exceeds the available ROM by several KB. Hence, the first step was to reduce the amount of required memory for the base application, i.e. TinyIPFIX, to make room for TinyECC. Since TinyIPFIX was designed to run with TinyOS 2.1.1, but the latest version 2.1.2 includes several optimizations regarding memory consumption and routing, the logical consequence was to upgrade the existing code to work with TinyOS 2.1.2. This entails modifications to the BLIP stack, which in turn requires the adaption of various

function calls. The result was an increase of available ROM of roughly 2 KB, Table 5.1 shows the measurements before and after the changes. Interestingly, the RAM requirements did increase with the OS migration, probably due to the changed routing protocol, which requires more caching. However, since the limiting factor in the present case is always the available ROM, this is not an issue.

| | **Version 2.1.1** | **Version 2.1.2** |
|---|---|---|
| Routing | Hydro | RPL |
| BLIP stack | 1.0 | 2.0 |
| RAM (Collector) | 6411 KB | 6976 KB |
| ROM (Collector) | 39480 KB | 37598 KB |

Table 5.1: Changes from TinyOS 2.1.1 to 2.1.2.

### 5.2.1 Handshake

The handshake, as it was designed in Section 4.3, is best implemented in a dedicated component. In the current implementation, this component is called `HandshakeHandler` and is exclusively responsible for the handling of handshake messages. This includes the composition of new messages as well as the handling of replies. `HandshakeHandler` is wired to other components that are called for the cryptographic functions. The three handshake messages from the protocol design are implemented as follows, where `nx_uint8_t` stand for the network-serializable unsigned integer type:

```
// Handshake 1: mote -> base station
typedef nx_struct to_hsm1_t{
    nx_uint8_t msgType;
    nx_uint8_t hsType; // 0x01
    nx_uint8_t public_x[24];
    nx_uint8_t public_y[24];
    nx_uint8_t digest[20];
} to_hsm1_t;

// Handshake 2: base station -> mote
typedef nx_struct to_hsm2_t{
    nx_uint8_t msgType;
    nx_uint8_t hsType; // 0x02
    nx_uint8_t public_x[24];
    nx_uint8_t public_y[24];
    nx_uint8_t digest[20];
    nx_uint8_t signature[]; // variable length, ANS.1 encoded
} to_hsm2_t;
```

```
// Handshake 3: mote -> base station
typedef nx_struct to_hsm3_t{
    nx_uint8_t msgType;
    nx_uint8_t hsType; // 0x03
    nx_uint8_t r[24];
    nx_uint8_t s[24];
} to_hsm3_t;
```

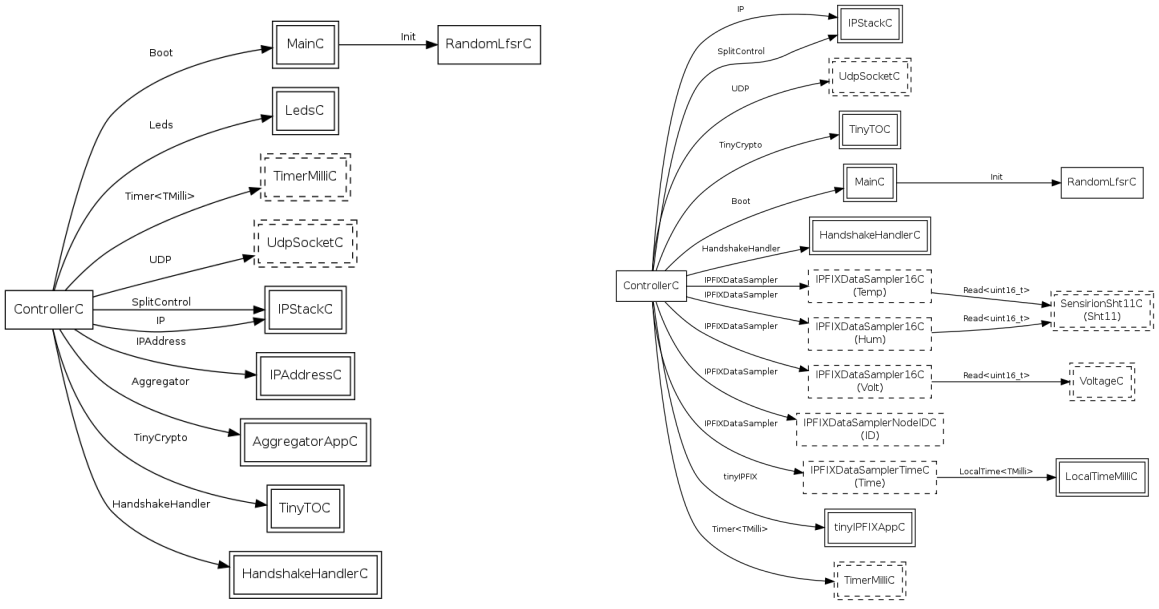
The `msgType` field is internally used to differentiate between handshake messages and other types of control messages that are sent via the same port. `hsType` is intuitively used to identify the different handshake messages, from number one to three. Further, public ECC keys are decomposed into x and y-coordinates for easy handling on the mote side. ECDSA signatures are somewhat difficult to include in a fixed-length packet, since the bit-length of `r` and `s` may vary, due to the hexadecimal representation of large integers. While it is not difficult to detect the actual length and process the signature accordingly, the TinyECC library is very selective with input parameters and requires accurate length information before signature validation. Thus, the signature in the second handshake message, i.e. from the station back to the mote, is encoded in the Abstract Syntax Notation One (ANS.1), which inherently includes length information for `r` and `s`. The reply, however, can be sent with two plain fixed-length arrays for the signature components, as the gateway can easily run the additional computations that are necessary to strip any padding and to correctly encode the signature. With those two different signature representation, the computation time on the mote is minimized. Similarly to the three handshake messages, the two necessary messages for the authentication of Aggregators to Collectors are implemented as follows:

```
// Aggregator Verification 1: mote -> basestation
typedef nx_struct to_av1_t{
    nx_uint8_t msgType;
    nx_uint8_t hsType; // 0x01
    nx_uint8_t address[16];
} to_cm1_t;

// Aggregator Verification 2: basestation -> mote
typedef nx_struct to_av2_t{
    nx_uint8_t msgType;
    nx_uint8_t hsType; // 0x02
    nx_uint8_t public_agg_x[24];
    nx_uint8_t public_agg_y[24];
    nx_uint8_t signature[]; // variable length, ANS.1 encoded
} to_cm2_t;
```

### 5.2.2 Cryptography

Similar to the handshake, the cryptography algorithms can be implemented in a dedicated component. The component developed in this thesis is called `TinyTO` (TO stands for two-way authentication optimization), and wired to TinyECC, which implements the actual computations in an efficient way. In the concrete implementation, `TinyTO` is responsible for keys generation, all key exchange related calculations, signature generation, signature validation, and encryption.
TinyECC offers various optimization techniques, which can increase the speed significantly. However, a drawback of all optimizations is the growing memory footprint. Thus, every optimization can be enabled or disabled individually, matching the available resources. Table 6.2 shows the memory consumption of optimizations and which optimizations are enabled in Aggregator and Collector nodes for the current implementation. Note, however, that every optimization can be toggled by just commenting or uncommenting one single line at compilation time, so this aspect can be adapted immediately for more powerful devices.
Wiring the existing solution, handshake, and cryptography components together results in a modular application, as it is shown in Figure 5.3, which contains the auto-generated component graphs of Aggregator and Collector. The notation is the same here as for all following component graphs: A single box denotes a module, a double box a configuration, dashed boxes generic components, and an ellipse an interface. Lines denote wiring, the description is the applied variable name in the implementation. Both new components can be used without any individual adaption in both applications, due to their generic design.




(a) Aggregator (b) Collector

Figure 5.3: Component graph of the TinyOS Applications.

As explained above, the two components that were developed in this thesis are `TinyTO` and `HandshakeHandler`. While one can see in Figure 5.3 the modular composition of

an application, the detailed component graph for both security components is shown in Figure 5.4. Both the `HandshakeHandler` and the `TinyTO` cryptography component are wired directly into the `Controller` application, allowing it to start key generation and handshake execution independently. The handshake handler necessarily accesses the cryptography component to retrieve ECC keys, to ask for signature generation, and signature validation. The underlying ECC algorithms (ECC, ECDSA, ECIES, and ECDH) are part of the TinyECC library, but wired individually into the `TinyTO` component.

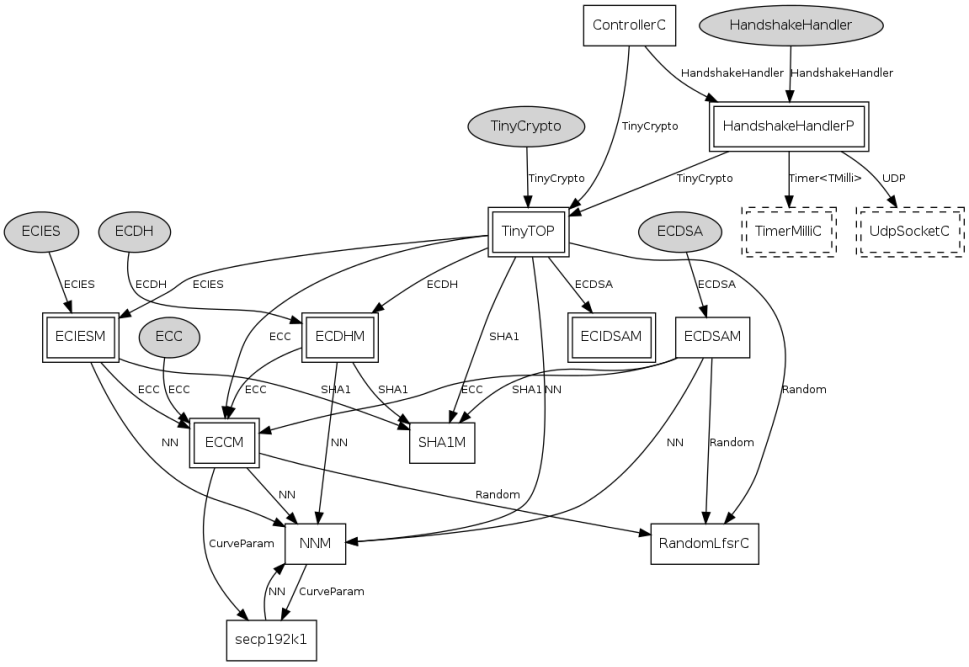



Figure 5.4: Wiring of security components.

## 5.3 Gateway

On the Gateway side, the WSN framework CoMaDa is used on a laptop computer [89]. CoMaDa is implemented in Java and provides a graphical user interface (GUI) for mote programming, WSN administration, and data presentation. It receives and processes the encrypted packets from a connected WSN. The complete gateway is formed by the laptop with CoMaDa and a base station mote (cf. Section 2.2), which is programmed with the TinyOS application PppRouter. The base station transmits all received packets to the serial port of the laptop, where the Linux tool pppd is used to forward them to CoMaDa. Since the Java implementation on a powerful laptop allows for native IPv6 communication without any abstraction like in BLIP, the PppRouter directly forwards the recovered IPv6 packets.

The security solution here is also implemented on the application level and further integrated into CoMaDa. Conceptually, it is located on the same position as in TinyOS: The application receives the data packet, uses the security component for decryption of the payload, and afterwards processes the original message.

As Java security provider, the free and open source library Bouncy Castle (BC) is used in version 1.50 (see [77] for more information). The general architecture of BC divides it into two components: a collection of different cryptographic Application Programming Interface (APIs) for Java and a Java Cryptography Extension (JCE) provider, which uses the lower APIs to implement cryptographic schemes. The JCE is an official standard to the Java Platform, allowing for a hight portability and easy integration [106]. Further, the BC JCE includes various cryptographic schemes for almost every common cryptographic approach, allowing for easy extension and modification of individual components, for example using a different hash function or key derivation function.

# Chapter 6

# Evaluation

This chapter contains the practical and theoretical evaluation of the presented security and handshake design. First, the experimental setup is described, then measurements of memory consumption and execution times on the testbed are performed. Further, the energy consumption of individual operations and radio transmissions is estimated to calculate the expected runtime of motes that are programmed with the proposed security implementation. Lastly, the security aspects and resulting limitations of the current implementation are evaluated.

## 6.1 Setup

The testbed consists of ten TelosB CM5000-SMA (see [112] for vendor information) sensor nodes, like the one shown in Figure 2.1. All of the motes are equipped with a slot for batteries (2x AA batteries), external 5 dBi SMA antenna, and sensors for temperature, relative humidity, and light. Further, they can be programmed and powered over a USB connector.
Given the classification of constrained nodes in Table 2.2, the TelosB platform is a borderline case of a class 1 device. While the 48 KB programmable ROM is well within the definition of a class 0 device ($\ll$100 KiB), the 10 KB RAM is more than usually to be expected from the smallest device class (which would be $\ll$10 KiB). However, as noted in Section 2.1.1, the numbers ultimately are only to be used as a guideline, and the TelosB platform is capable of handling a RFC-compliant IP stack for direct communication with the Internet. Therefore, it effectively falls into the device class 1. The chosen target platform for this thesis is TelosB, as it provides several desirable properties:

- TelosB motes are a lower device class. Whatever implementation can run on such a highly constrained device, will most likely work as well in an inhomogeneous environment with devices of higher class.
- Onboard sensors allow to simulate real-world applications, effectively proving that not only a theoretical solution is provided, but it can be integrated in real applications.

- Powerful radio communication allows for tests on spatially distributed motes, again simulating applications that are closer to the real world.

The data streams in the testbed are directed towards one predefined data sink. This is the *Gateway* and, as outlined in Section 2.2, a combination of a dedicated TelosB mote that acts as base station and a laptop computer that is forwarded all signals over the serial port. The laptop is further used to program all motes with the pre-shared keys and runs the CoMaDa application with all newly developed cryptographic components. For testing purposes, three Aggregators were deployed, each with two Collectors, resulting in a total of nine deployed nodes.

## 6.2 Memory Consumption

The main challenge of this thesis was to design a security solution that requires only a small part of the available resources, allowing for a reasonable application to be running in addition to the security solution. Thus, the following sections will report the results of the evaluation concerning memory and energy consumption, which is calculated based on the performance measurements.

The memory consumption of applications can be estimated for TinyOS directly with the output of the make tool. All resources, as well as the component graph, are already known at compile-time, as nesC is a static language. To further record the memory footprint of specific components, it is often possible to deactivate some of them via the make file (e.g. RPL and TinyECC optimizations). In other cases, one can compile the complete application and then to gradually remove individual components (e.g. the TinyTO component and the handshake handler). Table 6.1a shows the memory consumption of the most important components in the Collector application, Table 6.1b of components in the Aggregator application.
The small difference in size between conceptually identical components in both implementation is a result from marginally different use-cases. For instance, Collectors need to store only one message at a time. Aggregators, however, need to reserve additional space to cache messages from every Collector before aggregation can be performed. Since memory is statically reserved, the memory footprint, thus, depends on the degree of aggregation. Further, Collectors only need the code to encrypt messages, whereas Aggregators need to both decrypt incoming and encrypt outgoing messages. On the other hand, Collectors are wired to periodically read their sensor values, which is not part of the Aggregator application. All in all, this leads to a memory consumption of 37590 byte purely for data collection and 33174 byte purely for data aggregation (cf. Figure 6.1a and 6.1b), yielding a bit over 4 KB of additional free memory in Aggregators. This memory can be used to enable more ECC optimizations, where Collectors are already at their memory limit. The memory requirements for individual techniques can be found in Table 6.2, where the activated optimizations are marked. How the difference impacts the resulting performance for ECC operations on Aggregators and Collectors is subject of Section 6.3.

| | **ROM [byte]** | **RAM [byte]** |
|---|---|---|
| Handshake | 1138 | 612 |
| Cryptography | 9378 | 406 |
| **TinyTO Total** | 10516 | 1018 |
| Data Collection | 31344 | 5478 |
| RPL | 6228 | 1498 |
| **Collector Total** | 37598 | 6976 |
| **Collector with TinyTO** | **48114** | **7994** |

(a) Collector

| | **ROM [byte]** | **RAM [byte]** |
|---|---|---|
| Handshake | 1636 | 602 |
| Cryptography | 11406 | 406 |
| **TinyTO Total** | 13042 | 1018 |
| Data Aggregation | 26904 | 6964 |
| RPL | 6270 | 498 |
| **Aggregator Total** | 33174 | 7462 |
| **Aggregator with TinyTO** | **46216** | **8470** |

(b) Aggregator

Table 6.1: Memory consumption of different components.

| | **ROM [Byte]** | **RAM [Byte]** | **Aggregator** | **Collector** |
|---|---|---|---|---|
| Barrett reduction | 780 | 114 | - | - |
| Hybrid multiplication | 12 | 0 | X | - |
| Hybrid square | 114 | 0 | X | X |
| Secpt optimization | 414 | 0 | X | - |
| Projective coordinates | 850 | 0 | X | X |
| Sliding window | 206 | 2350 | - | - |

Table 6.2: Memory consumption of different TinyECC optimizations and their utilization in the security components.

## 6.3 Performance

In terms of performance, the slow microcontroller and small memory size have an high impact on the results. All operations were evaluated with a message size of 116 byte, as it is typically generated in the underlying TinyIPFIX implementation for the TelosB platform, while Collectors naturally do not need to run decryption algorithms. The measurements were performed on the experimental setup with TelosB motes, where a timer is read before and after a operation was executed. The timer has a resolution of 65.53 milliseconds, hence the systematic error is ± 13 seconds for every measurement. Any potential bias in the timer cannot be determined reliably without another mode of measurements. The traditional stopwatch approach was deemed as unreliable to measure the absolute time, since too many additional error sources are introduced, such as the response time of LEDs or human error in starting and stopping the clock, which introduce an ever bigger uncertainty of measurements. The random error is determined by taking three measurements for every operation. The resulting measurement error is computed according to the established formulas (cf. Appendix A).

| | **Aggregator** [s] | **Collector** [s] |
|---|---|---|
| EC Key Generation | $4.77 \pm 0.14$ | $8.77 \pm 0.17$ |
| SHA-1 | $< 0.1$ | $< 0.1$ |
| ECDSA Sign | $5.14 \pm 0.19$ | $9.28 \pm 0.18$ |
| ECDSA Verifiy | $10.20 \pm 0.19$ | $18.51 \pm 0.19$ |
| ECIES Encrypt | $5.98 \pm 0.15$ | $9.41 \pm 0.18$ |
| ECIES Decrypt | $4.96 \pm 0.19$ | - |

Table 6.3: Execution time for individual ECC operations.

Table 6.3 shows the execution times for various cryptographic operations in Aggregators and Collectors. As one can see, the Aggregators are generally almost twice as fast as Collectors, since more ECC optimizations can be enabled (cf. Section. 6.2). However, Liu et al. give a performance evaluation in their paper for an identical hardware platform, that shows the speed for ECC operations when all optimizations are used: For instance, ECDSA signing then takes only about 1.6 seconds (Aggregators need ∼5.14 seconds, Collectors ∼9.28 seconds), verification about 2.0 seconds (Aggregators need ∼10.20 seconds, Collectors ∼18.51 seconds). This is obviously much faster than the presented application and really shows the performance limitations due to the restricted memory. The proposed solution could potentially run much faster with slightly more memory, even if the the platform does not change.
The resulting execution times of composite operations in the application can be found in Table 6.4. This includes transmission times in both directions and the response calculations from the Gateway. However, since the latter is extremely fast compared to motes, this does not impact the overall result much. The value for message aggregation with $doa = 2$ is calculated based on two message decryption operations and one message encryption operation on the Aggregator.

| | **Time [s]** |
|---|---|
| Handshake Aggregator | $20.89 \pm 0.18$ |
| Handshake Collector | $36.79 \pm 0.18$ |
| Aggregator verification | $19.44 \pm 0.19$ |
| Message aggregation (doa=2) | $15.90 \pm 0.53$ |

Table 6.4: Execution time for operations in the application.

Using the combined results from above, it is possible to determine the minimal interval $t$ in which Collectors may send their encrypted messages so that Aggregators can still catch up with the incoming packets: Once the handshake is executed, Collectors send data after max($t$, $(9.41 \pm 0.18)$ seconds), even if new data is created immediately, as that is the minimal time required to encrypt the message after data collection. Aggregators, on the other hand, require (doa $\cdot$ $(4.96 \pm 0.19)$ seconds ) + $(5.89 \pm 0.15)$ seconds to encrypt *doa* incoming messages and to encrypt the aggregated value. For a $doa \geq 2$, this gives the following guideline for the Collector's transmission interval:

$$t \geq doa \cdot 4.96\ seconds + 5.89\ seconds \tag{6.1}$$

It is to be noted, however, that this equation assumes enough memory to be available on the Aggregator side to cache *doa* packets, plus an immediate computation of aggregates. While this may hold for a small *doa*, the restriction should theoretically be kept in mind for larger networks. Although it can be assumed that an Aggregator, that is powerful enough to provide a much larger *doa* is also much faster in encryption and decryption, so the formula would need to be adapted.

### 6.3.1 Energy Consumption

A main aspect when determining the performance of a solution in WSNs is the power consumption. Typically, motes are equipped with batteries when in deployment, which are potentially difficult to replace when a mote is not easily accessible. As such, it is impractical to use a solution that requires a large amount of energy, as it could drain the battery too fast for practical purposes. It is assumed in the following evaluation that motes are powered by two off-the-shelf batteries, each with a capacity of 2000 mAh and voltage 1.5 V, in total delivering $U = 3.0\ V$.
The energy consumption of WSN motes can mainly be attributed to two components: 1) wireless data transmission and 2) computations in the microcontroller. Auxiliary components, such as LEDs or serial connectors are not taken into consideration, as they typically are not used in active deployment.

As outlined in Section 6.1, TelosB motes are used exclusively in the testbed. They are equipped with a Texas Instruments CC2420 RF chip for radio communications on the IEEE 802.15.4 2.4 GHz band (see [47] for vendor information). It is assumed that they transmit data with the maximum output power of 0 dBm. The current draw for this

transceiver is generally measured to be as high as $I_{Tx} = 17.4\ mA$ for sending (Tx) and $I_{Rx} = 19.7\ mA$ for receiving (Rx) of data [75, 87], which is slightly higher than the official data-sheet states. The theoretical transmission rate of the CC2420 is given in the same sources as 250 kbps, but some practical measurements are as low as 180 kbps. Hence, for the purposes of the calculation being as close to reality as possible, it can be assumed that the full transmission speed is never actually reached and thus $R = 220 \pm 20\ kbps$. Using the transmission rate and the implementation details of messages (cf. Section 5.2.1 and Section 3), one can easily compute how long the transmission of each message takes and from that the energy consumption per message. The corresponding formula to calculate the total energy $E_R$ depending on the message size $S$ for given voltage, current draw, and transmission rate is:

$$E_R(S) = U \cdot I \cdot \frac{S}{R} \tag{6.2}$$

In particular, this concerns the three handshake messages (denoted as HS 1 to HS 3, cf. Section 5.2.1), the Aggregator verification (denoted as AV 1 and AV 2, cf. Section 5.2.1), and the two types of IPFIX messages (templates and data). However, the messages as they are lined out in the handshake design (cf. Section 5.2.1) consider only the size of individual data fields in an unencrypted message. In reality, the transmitted packages on the IEEE 802.15.4 MAC layer are much larger: The available payload in a IEEE 802.15.4 message on the MAC layer is only 102 byte out of the total frame size of 127 byte (cf. Section 2.3). From those 102 byte, 12 byte are used by TinyOS and the Cyclic Redundancy Check (CRC) for error-detection. This leaves a total of 90 byte for the actual payload in every message on the MAC layer. Now, the handshake messages are not transmitted as plaintext, but larger ECIES ciphertext, requiring 79 byte (1 byte to indicate the point compression type, 24 byte for each EC point component, and 20 byte for message authentication) more than the pure message size. For example, the first handshake message HS 1 has a size of 70 byte, according to the implementation (cf. Section 5.2.1). Encrypted with ECIES, the size is, thus, 139 byte. Given the maximum payload size of 90 byte, the message has to be split in two parts of less than 90 byte each, in order to fit into MAC layer packets. Every part is again decorated with the different headers and other fields. Thus, the effective data size $DS$ that is transmitted to convey a payload of size $ps = 139\ byte$ has to be calculated as:

$$DS = ps + \lceil \frac{ps}{90\ byte} \rceil \cdot 37\ byte \tag{6.3}$$

$$DS = 139\ byte + \lceil \frac{139\ byte}{90\ byte} \rceil \cdot 37\ byte = 213 byte \tag{6.4}$$

Table 6.5 shows the results of these considerations for the different message types.

Similar to the energy calculations for radio transmissions, one can compute the energy consumption of the microcontroller for different cryptographic operations. The built in microcontroller on TelosB motes is the MSP430F1611 (see [46] for vendor information) 16 bit Ultra-Low-Power MCU (microcontroller unit) from Texas Instruments. The current draw in active mode (i.e. only MCU, no radio transmissions) is $I_{AM} = 1.8\ mA$ [81, 75]. The formula used to calculate the energy consumption $E_{MCU}$ of the MCU depending on the computation time $t$ subsequently is:

$$E_{MCU}(t) = U \cdot I_{AM} \cdot t \tag{6.5}$$

| Message | ps [byte] | DS [byte] | Time [ms] | Energy [mJ] |
|---|---|---|---|---|
| HS 1 (Tx) | 139 | 223 | $8.11 \pm 0.74$ | $0.42 \pm 0.04$ |
| HS 2 (Rx) | 189 | 300 | $10.91 \pm 0.99$ | $0.64 \pm 0.06$ |
| HS 3 (Tx) | 114 | 203 | $7.38 \pm 0.67$ | $0.38 \pm 0.03$ |
| AV 1 (Tx) | 82 | 171 | $6.22 \pm 0.56$ | $0.32 \pm 0.03$ |
| AV 2 (Rx) | 168 | 242 | $8.80 \pm 0.80$ | $0.52 \pm 0.05$ |
| IPFIX template (Tx) | 180 | 254 | $9.24 \pm 0.84$ | $0.48 \pm 0.04$ |
| IPFIX data (Tx) | 148 | 222 | $8.07 \pm 0.73$ | $0.42 \pm 0.03$ |
| IPFIX template (Rx) | 180 | 254 | $9.24 \pm 0.84$ | $0.54 \pm 0.05$ |
| IPFIX data (Rx) | 148 | 222 | $8.07 \pm 0.73$ | $0.47 \pm 0.04$ |

Table 6.5: Energy consumption of radio transmissions.

| | Aggregator | | Collector | |
|---|---|---|---|---|
| **Operation** | **Time [s]** | **Energy [mJ]** | **Time [s]** | **Energy [mJ]** |
| EC Key Generation | $4.77 \pm 0.14$ | $8.59 \pm 0.03$ | $8.77 \pm 0.17$ | $15.78 \pm 0.13$ |
| ECDSA Sign | $5.14 \pm 0.19$ | $9.25 \pm 0.19$ | $9.28 \pm 0.18$ | $16.70 \pm 0.16$ |
| ECDSA Verify | $10.20 \pm 0.19$ | $18.36 \pm 0.19$ | $18.51 \pm 0.19$ | $33.32 \pm 0.19$ |
| ECIES Encrypt | $5.98 \pm 0.15$ | $10.76 \pm 0.06$ | $9.41 \pm 0.18$ | $16.94 \pm 0.16$ |
| ECIES Decrypt | $4.96 \pm 0.19$ | $8.93 \pm 0.19$ | - | - |

Table 6.6: Energy consumption of different cryptographic operations.

Table 6.6 shows the results of the calculations. Due to the different ECC optimizations in Aggregators and Collectors, the results differ for both roles. Given the cost of radio transmission and computation of single cryptographic operations, it is then easy to calculate the energy consumption for the whole handshake and similar more complex sequences of operations. According to the design, as previously shown in Figure 4.6, the handshake requires the following operations:

- EC Key Generation
- sending of HM 1
- reception of HM 2
- ECDSA signature verification
- ECDSA signature signing
- sending of HM 3

Similarly, the verification of an Aggregator needs three operations, as shown in Figure 4.8:

- sending of the Aggregator identity in AV 1
- reception of the signed message containing the public key in AV 2
- ECDSA signature verification

And finally the aggregation for $doa = 2$ on the Aggregator requires a combination of operations as follows:

- reception of two IPFIX data packets
- two times ECIES decryption
- ECIES encryption
- sending of IPFIX data packet

The results of the corresponding calculations can be found in Table 6.7.

| **Operation** | **Time [s]** | **Energy [mJ]** |
|---|---|---|
| Handshake Aggregator | $20.14 \pm 0.52$ | $37.64 \pm 0.54$ |
| Handshake Collector | $36.59 \pm 0.54$ | $56.02 \pm 0.62$ |
| Aggregator verification | $18.52 \pm 0.19$ | $34.16 \pm 0.27$ |
| Message aggregation (doa=2) | $15.90 \pm 0.53$ | $30.12 \pm 0.57$ |

Table 6.7: Energy consumption of composite operations.

Given the configuration of batteries in motes, where individual cells are connected in serial, the capacity of a combined battery pack is 2000 mAh with a voltage of 3 V. TelosB motes require a minimal voltage of 1.8 V to stay functional, meaning a battery cannot be depleted to an energy level below 60 % of the original charge, otherwise the voltage will drop below the threshold. Thus, one can assume that 12.96 kJ of energy are available in one set of batteries. If Collectors are programmed to collect, encrypt, and send data every interval $t$, the expected runtime of Collectors and their Aggregators can be easily calculated, as it is shown in Table 6.8. It is assumed that every $10^{th}$ transmission contains the template instead of data values and Aggregators aggregate the values of two Collectors. Further, the motes are in sleep mode between measurements, which might change for other applications.

| Interval $t$ | Aggregator | Collector |
|---|---|---|
| 1 $min^{-1}$ | 7 h 15 min | 12 h 28 min |
| 1 $h^{-1}$ | 18 d | 31 d |
| 12 $h^{-1}$ | 217 d | 374 d |
| 1 $d^{-1}$ | 62 w | 107 w |

Table 6.8: Estimated runtime of motes with the proposed security solution.

## 6.4 Security

The security aspect of the proposed solution is a direct result from the threat analysis from Section 2.4. As such, most of the listed attacks can be countered and all of the security goals are met. In particular, the upgrade to BLIP 2.0 with RPL routing has solved many of the previous problems, as it improves routing with various measurements that can be used to detect and defend against different attacks (cf. Section 2.3 and Section 4.1). Further, the handshake design with mutual authentication adds immensely to the security level of the proposed solution (cf. Section 4.3). The direct and mutual authentication is currently only provided by solutions that either require additional infrastructure like certificate authorities (PAuthKey, TinyDTLS), cannot dynamically add new motes to the network (Merkle Trees), or run only on hardware platforms beyond class 1 (HIPDEX). Other solutions, such as the UbiSec&Sens approach with a Zero Common Knowledge approach, fulfill the requirements but are vulnerable to man-in-the-middle attacks. Table 6.9 shows the previously listed threads, security goals, and how they were addressed in the developed solution.

## 6.5 Limitations

The limitations of the proposed approach are, again, mostly dictated by the low resources of class 1 devices. In particular, Aggregators are so constrained in memory that they cannot aggregate more than two Collectors without a reduction of the ECC optimization, thus sacrificing most of the speed advantages. This gives a trade-off between aggregation speed and degree of aggregation, where the use-case determines the settings: Either to aggregate many data sources in very long intervals, or to aggregate only one or two sources more often. But even for a *doa* of two, the required time is much more than data aggregation in TinyIPFIX without security, which happens almost instantaneous. There exists no immediate solution on the software level, only more powerful devices (e.g. Opal motes) could be used to substitute critical Aggregators.
A subsequent drawback from the extensive computation time is the availability of Aggregator nodes. Due to the non-preemptive design of TinyOS components, a mote cannot queue packets upon receipt if another task is currently running. This means that data

| Issue | Solution |
|---|---|
| Network Layer security | RPL routing with BLIP 2.0 |
| Transport Layer security | Reverse multicast routing and limited degree of aggregation |
| Fabricated identity defense | Mutual authentication protocol with PSK |
| Stolen identity defense | Mutual authentication protocol with PSK |
| Sinkhole attack defense | RPL routing with BLIP 2.0 |
| Hello flood defense | Mutual authentication protocol with PSK |
| Man-in-the-Middle defense | Mutual authentication protocol with PSK |
| Eavesdropping defense | 192 bit ECIES encryption |
| Confidentiality | 192 bit ECIES encryption |
| Integrity | ECIES includes message authentication |
| Authenticity | Mutual authentication and secure key exchange |
| Availability | Aggregators signal when their aggregation limit is reached |
| Accountability | Gateway has knowledge of all actions in the network |
| End-to-End Security | 192 bit ECIES encryption |
| Support for unreliable Communication | UDP transmission |
| Two-way Authentication | Mutual authentication protocol with PSK |
| Elliptic Curve Cryptography | TinyECC implementation |

Table 6.9: Threads, security goals, and their solutions.

packets from Collectors are dropped if the Aggregator is encrypting or decrypting a message at this exact moment. Since the fastest possible encryption times in the experimental setup are in the range of 5 seconds, this happens easily when more than one Collector is connected to the Aggregator and both Collectors send messages within a 5 second window. This can be partially avoided by programming Collectors with large enough intervals and starting the transmissions more than 5 seconds apart. However, this phenomenon is not new to the implementation for this thesis, but was already existent in the underlying TinyOS application. Although due to the missing security aspects, and subsequently much shorter computation times, the probability of such a collision was low enough to stay unnoticed before. A potential solution here is the modification of the application to provide multi-threading, as proposed by Klues et al. in the TOSThreads package for TinyOS [54], although this will most likely inflict issues with the available memory. Otherwise it might be possible to implement a basic form of synchronization between motes, to avoid collisions at run-time.

Another limitation in terms of security is the key generation on motes. To calculate a unique key-pair, a random number is used as a seed for calculations. Typically, this seed is the current system time, or any other source of entropy. However, typical WSN mote platforms, like the TelosB motes in this thesis, do not provide any real-time clocks: They completely loose power once the battery is removed and to not contain a dedicated battery to keep a clock running, as it would be the case in a personal computer, thus restarting the clock with the rest of the application. Consequently, a deterministically executed application will always use the same seed at run-time to calculate the exact same key every time. TinyOS has avoided this partially by using the IPv6 suffix of a mote as default seed, since only one node at a time can get assigned one particular suffix. Unfortunately, this still allows to calculate the exact key-pair of a mote based on its IP address. The obvious solution is to use a completely different entropy source for the seed. In a personal computer this can be, for example, the user input. But since WSN motes do not offer any direct user interaction, this is unfeasible. The only varying conditions on sensor nodes is environmental data from the sensors, e.g. the current voltage of the energy source or temperature. However, using sensor data as a seed for a completely secure key generation requires the following assumptions to hold: a) environmental data is recorded in a very fine granularity that differs between even very similar motes in close proximity, b) the sensor readings cannot be reproduced by an attacker, and c) every mote in the network has access to a sensor. But none of those three points can be assumed for TelosB motes and the proposed implementation: a) readings like temperature do not always differ from a mote that is very close by, b) other readings, like voltage, can easily be reproduced if for instance a stable external energy source is used instead of batteries, and c) Aggregators cannot use their sensors in the current implementation. The only reasonable solution is hence to introduce some form of time synchronization for motes at start-up, for example using the flooding time synchronization protocol [68]. This could also be used to avoid the previously mentioned collisions between messages and offers a global time source that can seed different keys for every single startup.

The last limitation in the current solution is concerning the data transmission between a Collector and its Aggregator. Preceding mutual authentication between nodes and the Gateway is used to confirm the identity and public key of an Aggregator to the Collector upon announcement, which guarantees confidentiality and integrity of the message. Thus, it is impossible for an attacker to get access to sensitive information from Collectors. But in the other direction, the Aggregator currently does not explicitly verify the identity of his Collectors. Ideally, the Aggregator would check with the Gateway for verification of every single Collector (as Collector and Aggregator do not share a PSK) and then run another mutual authentication protocol and key exchange with the Collector. This, however, introduces an enormous amount of additional computations and memory requirements, as the Aggregator would need to store all additional key pairs. Although this is still a more suitable solution than simply distributing pre-shared keys before deployment (cf. Section 3, e.g. the CDA scheme), since it still allows networks to form dynamically and to add new nodes at runtime, it is currently not feasible to provide that functionality under the given hardware constraints. It is to be noted that this issue is only prevalent at Aggregators. The Gateway can naturally verify the validity of every Collector, so that a network without aggregation is completely secure. This alone sets the proposed solution apart from other approaches like TinySAM, where the authentication is only one-way in any case.

# Chapter 7

# Summary and Conclusions

Given the development of a security solution in this thesis and the overview of related approaches in the area of WSN security, one can draw several conclusions. For one, security is expensive in terms of memory and computations. While it is generally easy to implement security components for communication channels in the Internet on powerful machines, the recent trend of small interconnected devices in the IoT brings new challenges. The limited resources do not allow to transfer existing solutions to smaller devices, like WSN nodes, without severe restrictions. Thus, it is necessary to design new solutions specifically for security in the IoT and extremely resource constrained devices.
Many proposed solutions from the past require additional infrastructure and simply more powerful hardware platforms to overcome the limitations, or run on constrained devices simply as proof of concept with not enough resources for an actual program. Further, most solutions are either still vulnerable to common attacks like the man-in-the middle or sacrifice the flexibility of network layouts by pre-distributing a precisely calculated set of keys. Hence, the main contribution of of this thesis is the design of a security solution for WSN motes in the IoT that delivers two-way authentication in a three-way handshake, as well as end-to-end security of network communications for devices as small as class 1. It applies ECC operations, that require less memory and fewer computations than alternative approaches, and uses pre-shared masterkeys only for authentication towards the Gateway. Hence, the proposed approach offers a higher level of security than other designs, while guaranteeing complete flexibility of the network topology. It is absolutely feasible to program and deploy new motes at runtime, without compromising the security scheme. Additionally, the implementation of the design is so resource efficient that in can establish a secure connection in seconds, while enough memory is available to readily execute data collection and aggregation tasks. This sets the design apart from most other suggested solutions, where none of the comparable implementations can achieve this level of security and flexibility on that small devices without depending on infrastructure like certificate authorities.

Future extensions of the implementation may include several features that can counter the current limitations, as they were outlined in Section 6.5. This could include clock synchronization between motes, multi-threading, or further optimization in order to deploy even smaller devices, for instance IRIS motes.

# Bibliography


[1] M. Abdalla, M. Bellare, and P. Rogaway. DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem. *Standard Specifications For Public-Key Cryptography*, 1999. IEEE.

[2] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight Jr., R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous Computing. *Communications of the ACM*, 43(5):74–82, 2000.

[3] H. Alemdar and C. Ersoy. Wireless Sensor Networks for Healthcare: A Survey. *Computer Networks*, 54(15):2688 – 2710, 2010.

[4] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[5] M. Badra and I. Hajjeh. Key-exchange Authentication Using Shared Secrets. *Computer*, 39(3):58–66, March 2006.

[6] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *Transactions on Networking*, 5(6):756–769, Dec 1997.

[7] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management — Part 1: General. In *NIST Special Publication 800-57*, 2005.

[8] D. Basin, C. Cremers, and S. Meier. Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication. *Journal of Computer Security*, 21(6):817–846, 2013.

[9] A. Becher, Z. Benenson, and M. Dornseif. Tampering with Motes: Real-world Physical Attacks on Wireless Sensor Networks. In *Proceedings of the 3rd International Conference on Security in Pervasive Computing*, SPC, pages 104–118. Springer, 2006.

[10] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 419–428. ACM, 1998.

[11] S. Blake-Wilson, D. Johnson, and A. Menezes. *Key Agreement Protocols and Their Security Analysis*. Springer, 1997.

[12] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman Key Agreement Protocols. In *Selected Areas in Cryptography*, pages 339–361. Springer, 1999.

[13] S. Blake-Wilson and A. Menezes. Unknown Key-share Attacks on the Station-to-Station (STS) Protocol. In *Public Key Cryptography*, pages 154–170. Springer, 1999.

[14] S. Boeyen, T. Howes, and P. Richard. Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2. RFC 2559, IETF, April 1999.

[15] C. Bormann, M. Ersue, and A. Keranen. Terminology for Constrained-Node Networks. RFC 7228, IETF, May 2014.

[16] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003.

[17] M. Buettner, B. Greenstein, A. Sample, J. Smith, and D. Wetherall. Revisiting Smart Dust with RFID Sensor Networks. In *Proceedings of the 7th ACM Workshop on Hot Topics in Networks*, pages 37–42, 2008.

[18] S. Camtepe and B. Yener. Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks. In *Computer Security*, pages 293–308. Springer, 2004.

[19] R. Canetti and H. Krawczyk. Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology*, pages 453–474. Springer, 2001.

[20] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient Aggregation of Encrypted Data in Wireless Sensor Networks. In *Proceedings of the Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 109–117. ACM, July 2005.

[21] Q. Chang, Y. Zhang, and L. Qin. A Node Authentication Protocol Based on ECC in WSN. In *Proceedings of the International Conference on Computer Design and Applications*, volume 2, pages 606–609. IEEE, 2010.

[22] C. Chong and S. Kumar. Sensor Networks: Evolution, Opportunities, and Challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003.

[23] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, IETF, September 2013.

[24] Atmel Corporation. Datasheet AT90S/LS8535. `http://www.atmel.com/Images/doc1041.pdf`, 2001. Rev. 1041H–11/01.

[25] R. Cramer and V. Shoup. Design and Analysis of Practical Public-key Encryption Schemes Secure Against Adaptive Chosen Ciphertext Attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. ACM.

[26] H. Delfs and H. Knebl. *Introduction to Cryptography: Principles and Applications*. Information Security and Cryptography. Springer, 2007.

[27] J. Deng, R. Han, and S. Mishra. Intrusion Tolerance and Anti-traffic Analysis Strategies for Wireless Sensor Networks. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 637–646, June 2004.

[28] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic Third-party Data Publication. In *Data and Application Security*, pages 101–112. Springer, 2001.

[29] W. Diffie, P. Van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[30] Whitfield Diffie and Martin E Hellman. New Directions in Cryptography. *Transactions on Information Theory*, 22(6):644–654, 1976.

[31] W. Du, R. Wang, and P. Ning. An Efficient Scheme for Authenticating Public Keys in Sensor Networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc, pages 58–67. ACM, 2005.

[32] S. Duquennoy, G. Grimaud, and J. Vandewalle. The Web of Things: Interconnecting Devices with High Usability and Performance. In *Proceedings of the International Conference on Embedded Software and Systems*, pages 323–330, May 2009.

[33] B. Ertl. Data Aggregation using TinyIPFIX. *BSc Thesis at the Technical University Munich*, 2011.

[34] L. Eschenauer and V. Gligor. A Key-management Scheme for Distributed Sensor Networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47. ACM, 2002.

[35] D. Evans. The Internet of Things: How the Next Evolution of the Internet is Changing Everything. *CISCO white paper*, 1, 2011.

[36] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, IETF, June 1999.

[37] K. Gaarder and E. Snekkenes. Applying a Formal Analysis Technique to the CCITT X. 509 Strong Two-way Authentication Protocol. *Journal of Cryptology*, 3(2):81–98, 1991.

[38] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Sigplan Notices*, volume 38, pages 1–11. ACM, 2003.

[39] O. Gervasi. Computational Science And Its Applications. In *Proceedings of the International Conference on Computational Science and its Applications*. Springer, 2005.

[40] L. Gong. A Security Risk of Depending on Synchronized Clocks. *SIGOPS Operation Systems Review*, 26(1):49–53, January 1992.

[41] D. Hankerson, S. Vanstone, and A. Menezes. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[42] C. Haowen and A. Perrig. Security and Privacy in Sensor Networks. *Computer*, 36(10):103–105, Oct 2003.

[43] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The Platforms Enabling Wireless Sensor Networks. *Communications of the ACM*, 47(6):41–46, June 2004.

[44] W. Hu, H. Tan, P. Corke, W. Shih, and S. Jha. Toward Trusted Wireless Sensor Networks. *Transactions on Sensor Networks*, 7(1):5:1–5:25, August 2010.

[45] American National Standards Institute. ANS X9.62-2005: The Elliptic Curve Digital Signature Algorithm (ECDSA). *American National Standards Institute*, 2005.

[46] Texas Instruments. MSP430F161x MIXED SIGNAL MICROCONTROLLER datasheet. `http://www.ti.com/lit/ds/symlink/msp430f1611.pdf`, 2011. Accessed: 2014-08-18.

[47] Texas Instruments. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. `http://www.ti.com/lit/ds/symlink/cc2420.pdf`, 2014. Accessed: 2014-08-18.

[48] ISO/IEC. International Standard 7498-1. ISO/IEC, International Organization for Standardization, 1994.

[49] Y. Jeong and S. Lee. Hybrid Key Establishment Protocol Based on ECC for Wireless Sensor Network. In *Ubiquitous Intelligence and Computing*, pages 1233–1242. Springer, 2007.

[50] M. Johnson, M. Healy, P. van de Ven, M. Hayes, J. Nelson, T. Newe, and E. Lewis. A Comparative Review of Wireless Sensor Network Mote Technologies. In *Sensors*, pages 1439–1442. IEEE, 2009.

[51] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brünig. Opal: A Multiradio Platform for High Throughput Wireless Sensor Networks. *IEEE Embedded Systems Letters*, 3(4):121–124, 2011.

[52] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2007.

[53] C. Karlof and D. Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Ad Hoc Networks*, 1(2):293–315, 2003. Elsevier.

[54] K. Klues, C. Liang, J. Paek, R. Musaloiu-Elefteri, P. Levis, A. Terzis, and R. Govindan. TOSThreads: Thread-Safe and Non-invasive Preemption in TinyOS. In *Proceedings of the Conference on Embedded Networked Sensor Systems*, volume 9, pages 127–140. ACM, 2009.

[55] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis. Evaluating the Performance of RPL and 6LoWPAN in TinyOS. In *Workshop on Extending the Internet to Low Power and Lossy Networks*. ACM, 2011.

[56] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart Objects as Building Blocks for the Internet of Things. *Internet Computing*, 14(1):44–51, Jan 2010.

[57] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle. DTLS Based Security and two-way Authentication for the Internet of Things. *Ad Hoc Networks*, 11(8):2710 – 2723, 2013.

[58] B. Krishnamachari, D. Estrin, and S. Wicker. The Impact of Data Aggregation in Wireless Sensor Networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 575–578, 2002.

[59] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919, IETF, August 2007.

[60] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger Security of Authenticated Key Exchange. In *Provable Security*, pages 1–16. Springer, 2007.

[61] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, and E. Brewer. TinyOS: An Operating System for Sensor Networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.

[62] F. Lewis. Wireless Sensor Networks. *Smart Environments: Technologies, Protocols, and Applications*, pages 11–46, 2004.

[63] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421, IETF, February 1993.

[64] A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks*, pages 245–256. IEEE, 2008.

[65] Y. Liu, J. Li, and M. Guizani. PKC Based Broadcast Authentication using Signature Amortization for WSNs. *Transactions on Wireless Communications*, 11(6):2106–2115, 2012.

[66] P. Lowack. Key Management in Wireless Sensor Networks with Support for Aggregation Nodes. *MSc Thesis at the Technical University Munich*, 2013.

[67] F. Mármol and G. Pérez. Security Threats Scenarios in Trust and Reputation Models for Distributed Systems. *Computers & Security*, 28(7):545–556, 2009. Elsevier.

[68] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 39–49. ACM, 2004.

[69] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC press, 2010.

[70] R. Min, M. Bhardwaj, S. Cho, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan. Low-power Wireless Sensor Networks. In *Proceedings of the 14th International Conference on VLSI Design*, pages 205–210, 2001.

[71] H. Modares, R. Salleh, and A. Moravejosharieh. Overview of Security Issues in Wireless Sensor Networks. In *Proceedings of the 3rd International Conference on Computational Intelligence, Modelling and Simulation*, pages 308–311, Sept 2011.

[72] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, IETF, September 2007.

[73] National and N. I. S. T. Technology. *Recommended Elliptic Curves for Federal Government Use*, 1999.

[74] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 259–268. ACM, 2004.

[75] H. Nguyen, A. Forster, D. Puccinelli, and S. Giordano. Sensor Node Lifetime: An Experimental Study. In *Pervasive Computing and Communications Workshops*, pages 202–207. IEEE, 2011.

[76] P. Nie, J. Vähä-Herttua, T. Aura, and A. Gurtov. Performance Analysis of HIP Diet Exchange for WSN Security Establishment. In *Proceedings of the 7th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pages 51–56. ACM, 2011.

[77] Legion of the Bouncy Castle Inc. The Legion of the Bouncy Castle. `http://www.bouncycastle.org`, 2013. Accessed: 2014-08-18.

[78] G. Padmavathi and D. Shanmugapriya. A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks. *Computing Research Repository*, 2009.

[79] A. Pathan, M. Alam, M. Monowar, and M. Rabbi. An Efficient Routing Protocol for Mobile Ad Hoc Networks with Neighbor Awareness and Multicasting. In *Proceedings of the E-Tech*, pages 97–100. IEEE, July 2004.

[80] A. Pathan, H. Lee, and C. Hong. Security in Wireless Sensor Networks: Issues and Challenges. In *Proceedings of the 8th International Conference on Advanced Communication Technology*, volume 2, pages 6 pp.–1048, Feb 2006.

[81] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-low Power Wireless Research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pages 364–369. IEEE, April 2005.

[82] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, and M. Ylianttila. PAuthKey: A Pervasive Authentication Protocol and Key Establishment Scheme for Wireless Sensor Networks in Distributed IoT Applications. *International Journal of Distributed Sensor Networks*, 2014. Hindawi Publishing Corporation.

[83] B. Preneel. Analysis and Design of Cryptographic Hash Functions, 1993.

[84] J. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Designing Privacy Enhancing Technologies*, pages 10–29. Springer, 2001.

[85] P. Rogaway and T. Shrimpton. Cryptographic Hash-function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-preimage Resistance, and Collision Resistance. In *Fast Software Encryption*, pages 371–388. Springer, 2004.

[86] K. Romer and F. Mattern. The Design Space of Wireless Sensor Networks. *IEEE Wireless Communications*, 11(6):54–61, 2004.

[87] C. Sadler and M. Martonosi. Data Compression Algorithms for Energy-constrained Devices in Delay Tolerant Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 265–278. ACM, 2006.

[88] M. Saleh and I. Al Khatib. Throughput Analysis of WEP Security in Ad Hoc Sensor Networks. In *Proceedings of the 2nd International Conference on Innovations in Information Technology*, pages 26–28, 2005.

[89] C. Schmitt, A. Freitag, and G. Carle. CoMaDa: An Adaptive Framework with Graphical Support for Configuration, Management, and Datahandling Tasks for Wireless Sensor Networks. In *Proceedings of the 9th International Conference on Network and Service Management*, pages 211–218. IEEE, Oct 2013.

[90] C. Schmitt, T. Kothmayr, B. Ertl, W. Hu, L. Braun, and G. Carle. TinyIPFIX: An Efficient Application Protocol For Data Exchange In Cyber Physical Systems. *Computer Communications*, 2014. Elsevier.

[91] J. Sen. A Survey on Wireless Sensor Networks Security. *International Journal of Communication Networks and Information Security*, 1(2):55 – 78, 2009.

[92] R. Shirey. Internet Security Glossary, Version 2. RFC 4949, IETF, August 2007.

[93] V. Shoup. A proposal for an ISO Standard for Public Key Encryption (version 2.1). *IACR E-Print Archive, ISO/IEC JTC 1/SC 27*, 112, 2001.

[94] V. Singh, S. Jain, and J. Singhai. Hello Flood Attack and its Countermeasures in Wireless Sensor Networks. *International Journal of Computer Science*, 7(11):23–27, 2010.

[95] F. Sivrikaya and B. Yener. Time Synchronization in Sensor Networks: A Survey. *Network*, 18(4):45–50, 2004. IEEE.

[96] IEEE Computer Society. IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Technical Report IEEE Std 802.15.4-2006, 2006.

[97] J. Solinas. Efficient Arithmetic on Koblitz Curves. *Designs, Codes, and Cryptography*, 19(2-3):195–249, March 2000. Kluwer Academic Publishers.

[98] Standards for Efficient Cryptography Group at Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. Standards for Efficient Cryptography Version 1.0. SECG2, 2000.

[99] G. Stoneburner. SP 800-33: Underlying Technical Models for Information Technology Security. Technical report, 2001.

[100] D. Surie, O. Laguionie, and T. Pederson. Wireless Sensor Networking of Everyday Objects in a Smart Home Environment. In *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks, and Information Processing*, pages 189–194, Dec 2008. IEEE.

[101] L. Wallgren, S. Raza, and T. Voigt. Routing Attacks and Countermeasures in the RPL-based Internet of Things. *International Journal of Distributed Sensor Networks*, 2013, 2013. Hindawi Publishing Corporation.

[102] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. Shantz. Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *Proceedings of the third International Conference on Pervasive Computing and Communications*, pages 324–328. IEEE, 2005.

[103] Y. Wang, G. Attebury, and B. Ramamurthy. A survey of security issues in wireless sensor networks. *Communications Surveys Tutorials*, 8(2):2–23, Feb 2006. IEEE.

[104] B. Warneke, M. Last, B. Liebowitz, and K. Pister. Smart Dust: Communicating with a Cubic-millimeter Computer. *Computer*, 34(1):44–51, 2001. IEEE.

[105] A. Weimerskirch and D. Westhoff. Zero Common-knowledge Authentication for Pervasive Networks. In *Selected Areas in Cryptography*, pages 73–87. Springer, 2004.

[106] J. Weiss. *Java Cryptography Extensions: Practical Guide for Programmers.* The Practical Guides. Elsevier, 2004.

[107] D. Westhoff, J. Girao, and M. Acharya. Concealed Data Aggregation for Reverse Multicast Traffic in Sensor Networks: Encryption, Key Distribution, and Routing Adaptation. *Transactions on Mobile Computing*, 5(10):1417–1431, 2006. IEEE.

[108] D. Westhoff, J. Girao, and A. Sarma. Security Solutions for Wireless Sensor Networks. *NEC Journal of Advanced Technology*, 59(2), 2006. IEEE.

[109] S. William and W. Stallings. *Cryptography And Network Security, 4/E.* Pearson Education, 2006.

[110] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, IETF, March 2012.

[111] D. Xiaojiang and C. Hsiao-Hwa. Security in Wireless Sensor Networks. *Wireless Communications*, 15(4):60–66, Aug 2008. IEEE.

[112] Advantic Sistemas y Servicios S.L. Advanticsys TelosB CM5000-SMA datasheet. `http://www.advanticsys.com/shop/mtmcm5000sma-p-23.html`, 2014. Accessed: 2014-08-18.

[113] C. Yibo, K. Hou, H. Zhou, H. Shi, X. Liu, X. Diao, H. Ding, J. Li, and C. De Vaulx. 6LoWPAN Stacks: A Survey. In *Proceedings of the 7th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4. IEEE, 2011.

[114] S. Zhu, S. Setia, and S. Jajodia. LEAP+: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks. *Transactions on Sensor Networks*, 2(4):500–528, 2006. ACM.

# Abbreviations

| | |
|---|---|
| 6LowPAN | IPv6 over Low power Wireless Personal Area Networks |
| A | Ampere |
| AES | Advanced Encryption Standard |
| ANS.1 | Abstract Syntax Notation One |
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| BC | Bouncy Castle |
| BLIP | Berkeley Low Power IP |
| BSL | Boot Strap Loader |
| CDA | Concealed Data Aggregation |
| CoAP | Constrained Application Protocol |
| CoMaDa | Configuration, Management, Data handling |
| CRC | Cyclic Redundancy Check |
| d | Day |
| dBi | Decibels-Isotropic |
| dBm | Decibel-Milliwatts |
| DES | Data Encryption Standard |
| DLP | Discrete Logarithm Problem |
| DOA | Degree Of Aggregation |
| DODAG | Destination Oriented Directed Acyclic Graph |
| DoS | Denial of Service |
| DSA | Digital Signature Algorithm |
| DTLS | Datagram Transport Layer Security |
| EC | Elliptic Curve |
| ECC | Elliptic Curve Cryptography |
| ECDH | Elliptic Curve Diffie-Hellman |
| ECDLP | Elliptic Curve Discrete Logarithm Problem |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| ECIES | Elliptic Curve Integrated Encryption Scheme |
| FIFO | First In First Out |
| h | Hour |
| HIP DEX | Host Identity Protocol Diet Exchange |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IoT | Internet of Things |
| IP | Internet Protocol |

| | |
|---|---|
| IPFIX | IP Flow Information Export |
| ISO | International Organization for Standardization |
| J | Joule |
| JCE | Java Cryptography Extension |
| JSON | JavaScript Object Notation |
| JTAG | Joint Test Action Group |
| KB | Kilobyte |
| KiB | Kibibyte |
| kbps | Kilobits per Second |
| KDF | Key Derivation Function |
| LED | Light-Emitting Diode |
| LLN | Low-Power Lossy Network |
| LowPAN | Low-Power Wireless Personal Area Network |
| LR-WPAN | Low-Rate Wireless Personal Area Network |
| mA | Milliampere |
| MAC | Media Access Control |
| mAh | Milliampere-hour |
| MCU | Microcontroller Unit |
| MHz | Megaherz |
| MITM | Man-In-The-Middle |
| mJ | Millijoule |
| ms | Millisecond |
| NIST | National Institute of Standards and Technology |
| OS | Operating System |
| PEM | Privacy Enhanced Mail |
| PHY | Physical Layer |
| PKC | Public Key Cryptography |
| PKE | Public Key Encryption |
| PKI | Public Key Infrastructure |
| PRNG | Pseudo Random Number Generator |
| PSK | Pre-Shared Key |
| RC4 | Rivest Cipher 4 |
| RFC | Request For Comments |
| RAM | Random Access Memory |
| ROM | Read Only Memory |
| RPL | Routing Protocol for Low power and Lossy Networks |
| RSA | Rivest, Shamir, Adleman |
| Rx | Receive |
| s | Second |
| SECG | Standards for Efficient Cryptography Group |
| SOAP | Simple Object Access Protocol |
| STS | Station-To-Station |
| SSL | Secure Socket Layer |
| TAP | Test Access Port |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| TPM | Trusted Platform Module |

| | |
|---|---|
| Tx | transmit |
| UDP | User Datagram Protocol |
| UKSA | Unknown Key-Share Attack |
| V | Volt |
| w | Week |
| WEP | Wired Equivalent Privacy |
| WoT | Web of Things |
| WPAN | Wireless Personal Area Network |
| WSN | Wireless Sensor Network |
| ZCK | Zero Common Knowledge |

# Glossary

**Aggregator** A WSN mote that is used to aggregate messages from different Collectors and transmit them to the sink. This can be either data aggregation, message aggregation, or both. It can collect sensor readings on its own, but that is not a requirement.

**Authentication** The act of confirming that someone is truthful about his identity. A protocol provides direct authentication if the authentication is complete with the end of a protocol run.

**Data Aggregation** Aggregators combine different data values into one aggregated value. For instance taking different temperature readings from sensors in the same room and calculating the average value as aggregate.

**Collector** A WSN mote that is used to collect and transmit sensor readings.

**Degree of Aggregation** The number of data values that are used to form one aggregate. Often abbreviated as *doa*.

**Elliptic Curve Cryptography** Public Key cryptography, where the structure of finite fields over elliptic curves is used to design a trapdoor function.

**Gateway** The Gateway connects the WSN to the Internet and routes messages between both. It is often composed of one dedicated sensor node as base station and a server that is connected to the Internet.

**Key Encapsulation** Using a symmetric key to encrypt a message and an asymmetric key on top of that to encrypt the symmetric key.

**Message Aggregation** Aggregators collect individual messages and transmit them in bulk. The message contents are unchanged and no data aggregration is performed on them.

**Mote** Another term for a node in a WSN, capable of performing processing and often equipped with one or more sensors to measure environmental data, e.g. temperature, humidity, or light.

**Sink** The sink is the target of data streams in a WSN. Usually the Gateway takes this role.

**Trapdoor Function** A mathematical function that is easy to compute, but difficult to invert.

# List of Figures

# List of Tables

# Appendix A

# Error Calculation

Assuming that $n$ values $x_1, x_2, ..., x_n$ are measured, the random error $\triangle\overline{x}$ is calculated as follows, where $t$ is the student-factor:

$$\overline{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \tag{A.1}$$

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (\overline{x} - x_i)^2} \tag{A.2}$$

$$\triangle\overline{x} = \frac{t \cdot \sigma}{\sqrt{n}} \tag{A.3}$$

The total error $E$ is calculated by adding the random error to the systematic error $\triangle\bar{t}$, which is determined beforehand:

$$E = \triangle\overline{x} + \triangle\bar{t} \tag{A.4}$$

Propagation of uncertainty is calculated differently, depending on the mathematical operation of uncertain values. For sums and differences, the resulting absolute error is simply the sum of both individual absolute errors. For division or multiplication of values $a$ and $b$, the relative error can be calculated as:

$$\frac{\triangle E}{E} = \frac{\triangle a}{a} + \frac{\triangle b}{b} \tag{A.5}$$

# Appendix B

# Installation Guidelines

This guideline shows how to build and deploy sensor motes with the TinyOS applications for data collections (Collector) and aggregation (Aggregator). The prerequisites are:

- Running Linux OS that supports TinyOS 2.1.2, for instance Ubuntu 12.04 or above.
- Installed TinyOS 2.1.2 toolchain (See `http://tinyos.stanford.edu/tinyos-wiki/index.php/Automatic_installation` for details).
- Installed Oracle JDK 1.7.0 or above.

To deploy a small WSN, with one Aggregator and two Collectors, do the following:

1. Open a Terminal at the source directory of the Collector code.
2. Connect a TelosB mote via the USB port. It is assumed to be USB0 in the following steps, use the command `motelist` to find out the actual port.
3. To program the mote as Collector with ID 1, type `make telosb blip install.1 bsl,/dev/ttyUSB0`.
4. Once the programming is completed, remove the mote from the port and connect the next mote.
5. To program this mote as second collector mote, repeat the first step with ID 2, i.e. type `make telosb blip install.2 bsl,/dev/ttyUSB0`.
6. Once the programming is completed, remove the mote from the port and connect the next mote.
7. Move into to source directory of the Aggregator code.
8. To program this mote as Aggregator with ID 3, similarly type `make telosb blip install.3 bsl,/dev/ttyUSB0`.

9. Once the programming is completed, remove the mote from the port and connect the next mote.

10. Move into the source directory of the PppRouter application, usually located in the TinyOS directory `/opt/tinyos-2.1.2/apps/PppRouter`.

11. Program the mote as base station by typing
`make telosb install.100 bsl,/dev/ttyUSB0`, but do not disconnect the mote once the programming is completed.

12. Run `sudo pppd debug passive noauth nodetach 115200 /dev/ttyUSB0 nocrtscts nocdtrcts lcp-echo-interval 0 noccp noip ipv6 ::23,::24`. Then open a different Terminal and type `ifconfig ppp0 add fec0::100/64`. The base station now forwards all packets to the ppp device.

13. Start the CoMaDa GUI to view the incoming messages and open the browser on `localhost:8000/index/protocols`.

14. Insert batteries into both Collectors. Once they appear in the GUI, insert batteries into the Aggregator.

# Appendix C

# Contents of the CD

- `Zusfsg.txt` : Unformatted text that contains the German abstract.
- `Abstract.txt` : Unformatted text that contains the English abstract.
- `Masterarbeit.pdf` : Complete thesis, including all appendices.
- `references/` : Collection of referenced papers, as possible.
- `code/TinyTOAggregation - BLIP 2.0.zip` : Complete source code for Aggregator and Collector motes with TinyTO.
- `code/wsn_framework.zip` : Complete source code for WSN framework CoMaDa with TinyTO.