Christian Zajc, BSc

Design and Implementation of an IoT-Security Demonstrator based on Contiki-OS

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Advisor: Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Dipl.-Ing. Dr.techn. Rainer Matischek (Infineon Technologies Austria AG)

Graz, November 2017

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present masters thesis.

Date	Signature

Abstract

In the last years, Internet of Things (IoT) became more and more popular. The main idea behind this technology is to improve the quality of life with intelligent devices, which provide smart behaviors, for all individuals. Diverse IoT environments, which are not recognized at first glance, are already distributed in products for homes. New products commonly provide a Wireless Fidelity (WiFi) interface for a typical home application in order to be able to connect the device to the Internet. With this opportunity the device can already be controlled with a smartphone inside the home or also from outside with the support of cloud services. Due to the increasing amount of these smart devices in our everyday life, an adequate security level has to be provided since they already collect and measure very sensitive data.

The sense for security is an important point in the IoT infrastructure. The usability of security features in products has to be as simple as possible so everyone will use it, however internally it still has to be complex to provide a high security level. This master thesis focuses on improving smart home applications with security features. These security features include the usage of hardware secured elements, hardware accelerated cryptographic functions and Near Field Communication (NFC) technology. Hardware secured elements in IoT devices provide the small nodes with hardware accelerated cryptographic functions and secured memory to envision an appropriate security level. NFC technology is used to improve the exchange of cryptographic keys between several IoT devices in a secured way. The only requirement for the usage of NFC technology is a typical smartphone, which is equipped with this technology. In addition to that, an Android application is developed in order to demonstrate the simple usability of exchanging cryptographic keys in a secured way by using Elliptic Curve Cryptography (ECC). This application can further be used for receiving status information of IoT nodes and for controlling these nodes by using NFC technology. Furthermore, the "Contiki Operating System (OS)" is used as base software component on the IoT devices, which is enhanced by several software modules.

The result of this master thesis is a demonstrator for a smart home application with enhanced security improvements. The smart home application is envisioned for making each home smart, meaning that the IoT nodes are designed to extend the original devices with intelligence. The communication between the IoT nodes is established with a radio unit and communicates in the Industrial, Scientific and Medical (ISM) frequency band of 868 MHz. In general, this wireless communication falls in the category of Wireless Sensor Networks (WSNs). Two end-user software platforms to control the smart home either with a website or an Android application are provided for each single user.

Kurzfassung

In den letzten Jahren etablierte sich Internet of Things (IoT) in den verschiedensten Anwendungsbereichen und wurde somit immer bekannter. Der Grundgedanke hinter dieser neuen Technologie ist, die Lebensqualität der Menschen mittels intelligenten Geräten zu verbessern. Bereits jetzt sind schon vielzählige IoT-Funktionalitäten in diversen Produkten im Haushalt integriert, die auf den ersten Blick nicht zu erkennen sind. Neue Produkte in diesen Bereichen beinhalten meistens ein Wireless Fidelity (WiFi)-Modul, welches eine Verbindung zum Internet herstellen kann. Damit können diese Geräte mittels eines Smartphones bedient werden. Durch die steigende Anzahl an IoT-Geräten im Alltag muss darauf geachtet werden, dass adäquate Sicherheitsmaßnahmen sichergestellt werden, denn bereits heute messen und speichern diese Geräte eine Vielzahl an sensiblen Daten.

In einer IoT-Infrastruktur ist es deshalb wichtig, ein besonderes Augenmerk auf Sicherheit zu legen. Die Verwendung von sicherheitsrelevanten Features in den Produkten sollte so einfach wie möglich gestaltet sein, damit jede Benutzerin bzw. jeder Benutzer diese verwenden kann, und dennoch intern so komplex, dass damit ein hoher Sicherheitslevel erreicht werden kann. Diese Masterarbeit fokussiert sich auf die Verbesserung einer Smart Home-Anwendung mit sicherheitsrelevanten Funktionen. Diese Funktionen umfassen die Verwendung von Hardware geschützten Elementen, hardwarebeschleunigten kryptographischen Algorithmen und Near Field Communication (NFC). Hardware geschützte Elemente werden in den IoT-Geräten verwendet, um kleine IoT-Knoten mittels kryptographischer Algorithmen und geschützten Speicherplatzes zu erweitern. NFC kommt zum sicheren und einfachen Austausch von kryptographischen Informationen für die Verschlüsselungen zum Einsatz. Damit der Benutzer diesen Schlüsselaustausch durchführen kann, wird lediglich ein Smartphone mit NFC-Funktion benötigt. Des Weiteren wird eine Android-Anwendung entwickelt, in der ein einfacher Schlüsselaustausch mittels Elliptic Curve Cryptography (ECC) durchgeführt werden kann. Mit dieser Anwendung können zusätzlich Funktionen ausgeführt werden, die den Erhalt von Statusinformation und die Steuerung der IoT-Geräte mittels NFC zur Verfügung stellen. Des Weiteren wird „Contiki Operating System (OS)“ als Softwarebasis für die IoT-Knoten eingesetzt, welches mit mehreren Software-Modulen erweitert wurde.

Das Ergebnis dieser Masterarbeit besteht aus einem Demonstrator für eine Smart Home-Anwendung, welche mit sicherheitsrelevanten Funktionen erweitert wurde. Die Smart Home-Anwendung ist so aufgebaut, dass jedes bestehende Gerät mittels des Einsatzes der entwickelten IoT-Knoten für IoT erweitert werden kann. Die Kommunikation zwischen den IoT-Knoten ist durch ein Funkmodul sichergestellt, welches im Industrial, Scientific and Medical (ISM) Frequenzband von 868 MHz arbeitet und sich somit in den Bereich von Wireless Sensor Network (WSN) einordnen lässt. Für den Endbenutzer stehen zwei unterschiedliche Software-Plattformen zur Verfügung, eine Webseite und eine Android Anwendung.

Acknowledgments

This master thesis was carried out at the Institute for Technical Informatics at the Technical University Graz. The practical part was executed at Infineon Technologies Austria in Graz. At first, I want to seize the chance to thank all people who supported me during the creation of the thesis and during my overall academic studies.

Especially, I would like to express my sincere gratitude to my supervisor Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger and to my advisor Dipl.-Ing. Dr.techn. Rainer Matischek for their continuous and professional support during the creation of this master thesis. Their contributed valuable feedback during this thesis sustainable improved the quality of the work. In addition, I want to thank all my colleagues at the "Cooperative Research and Exploration" department of the Infineon Development Center Graz for their great support and working atmosphere.

I also want to thank all my new friends, who I made during my time at the university, for making it such a gorgeous time and for the great experiences. I consider it as an honor to work with such an impressive and strong team of friends on a vast of projects and the team spirit at preparations of exams.

Finally, I want to express my very profound gratitude to my family for their support and patience during my studies, especially during the master thesis. I am very grateful for the unconditional support of my parents, Roman and Sabine Zajc, for their words of advice in difficult situations and financial support during my academic studies. They always supported me in all life situations to enable everything for me. In addition, I also want to thank my girlfriend, Elena Reich, for her support and patience. She helped me during difficult times with sustaining words. Thank you!

Graz, November 2017 Christian Zajc

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, Internet of Things (IoT) devices are becoming more and more popular in different areas. Typical fields of application are smart homes, health care, wearable devices, parking infrastructure, general public infrastructure and so on. A prediction of the amount of IoT devices by 2030 is estimated at 125 billion of installed devices [1]. One of the typical fields of application is smart homes or rather home automation, in order to introduce an increased usage of IoT devices. Generally, IoT applications either measure, monitor, or control multiplicity of parameters inside a smart home. These parameters cover in general following areas: monitoring air quality including temperature and humidity, controlling lights, managing door locks, controlling blinds, controlling diversity of electronic devices and similar actions. The introduction of smart devices inside a home or other places should increase the quality of life. In addition to the smartness, operational costs inside the field of application can also be reduced because the interconnectivity of several devices enables controlling procedures in smart and intelligent way.

1.1 Motivation

In consequence of increasing the number of deployed IoT devices in various fields of application, it is necessary to protect the privacy of each human who interacts with these devices. In the near future, each device inside a typical home will be connected to an IoT infrastructure. Each device generates a wide variation of sensitive data. With all this collected data, a central control unit can extract special features to control the connected devices in an intelligent interconnection with the combination of several devices.

In most IoT devices the importance of security issues are underestimated. Either they have a lightweight security architecture or they are not dealing with these topics. This master thesis is intended to provide security features for protecting the communication between exemplary IoT devices with an adequate security level. The security level and implemented security mechanism should be evaluated on their complexity and the possibility to implement the features on small embedded devices.

An additional perspective in the motivation of this work is to provide a simple way for the end-user to use security features in an IoT environment without having a deep understanding for the security topic. The reason for simplifying this procedure is that end-users only want to activate additional security features, when these steps are simple to proceed

and do not cost a lot of time. If the end-user struggles with the activation, then the security features will most probably not be used. Therefore, the usability of the process of activating security features should also kept in mind in the design process to be as simple as possible.

1.2 Objectives

The focus is lead on the development and implementation of a secured smart home application. The smart home application should be constructed with several IoT nodes, which extends the functionality of typical devices in a home such as outlets and switches. The process of extending typical devices introduces smartness to the objects. The main objectives of this thesis are to introduce a secured environment, which implies the encryption of all communication channels between the devices. An important point of applying security to the devices is the exchange of cryptographic keys in a deployed environment. Therefore, focus should be on using hardware secured elements with the support of Near Field Communication (NFC) technology to exchange the keys. The main activities and outcomes can be categorized into following points:

- State-of-the-art research of current IoT environments and technologies
- Develop and extend a smart home application with security improvements
- Improve the usability of the process of pairing between devices through the support of NFC technology
- Develop corresponding exemplary end-user software (website, Android application)
- Usage of hardware secured elements to secure the nodes also on low level

The outcome of this work is a demonstrator, which is demonstrating a typical smart home application. The importance of this smart home application is to provide a secured environment within the IoT devices. An additional key feature of this work should be to use NFC technology to exchange the cryptographic keys before the devices operates inside the IoT infrastructure.

1.3 Outline

Chapter 2 starts with a theoretical research of the master thesis and contains the investigations of the state-of-the-art products of the IoT environment. In addition, this Chapter also discusses topics of general Wireless Sensor Networks (WSNs), key management systems, typical network protocols for IoT environments and products on the market with focus on security flaws. Next, Chapter 3 describes the concept and design process of a secured smart home application. After the concept phase, Chapter 4 contains detailed information of the implementation of the previously described concept. Chapter 5 focuses on the resulted demonstrator for the smart home application with focus on a secured key exchange between the IoT devices. This chapter describes the resulting applications and the usability of providing a secured environment. At last, Chapter 6 summarizes the entire master thesis and provides a future outlook of possible enhancements.

Chapter 2

Related Work

This chapter provides an overall overview of the literature for the main topics, IoT and WSNs. At the beginning, the literature review focuses on the different environments of IoT and WSNs. After these topics, the state-of-the-art communication protocols are investigated in detail with their security flaws and possible attacks. Next, some basic key management concepts are described for miscellaneous network architectures. Finally, this chapter includes subjects about common products on the market, embedded Operating Systems (OSs), NFC, Java Card, and hardware security controller.

2.1 IoT (Internet of Things)

Nowadays, the modern term of IoT is commonly not only used in the field of wireless applications, but has a wide variation of definitions in the literature. The recap of these various definitions leads to the conclusion that IoT is not only one big technology but rather a combination of diverse technologies [2]. The process of combining various technologies follows one main goal. This goal has the task to enhance each device with smartness, inside a special environment, for example in a home, parking area, industry, or similar. In this context, the smartness means to establish a communication between all devices in order to process additional information and to make cross-device dependencies. With this additional information it is possible to interact with other devices and to control processes with services like IF-This-Then-That. Consequently with this service, an opened window can deactivate, for example, an air conditioner. However, to cope with a lot of various devices it is necessary to combine a wide spread of miscellaneous technologies like NFC, WSN with sensor or actuator nodes, Wireless Fidelity (WiFi), Ethernet, Internet connectivity, and similar technologies in an IoT environment.

An example for a smart integration, in a real life application, is to introduce smartness in a typical home. Each device inside a home can be extended with intelligent modules, which measures special variables of the environment. These variables can be parameters like temperature, power consumption, and object status. Consequently, each light bulb or outlet can be switched on and off by several other devices, which have the permission to do so. Additional devices like fridge, washing machine, heating, air conditioner and much more can be connected together to combine their added intelligence to reduce operational cost through the usage of an IF-This-Then-That logic. The smart control of processes are

possible through the additional knowledge of various parameters.

Typical fields of application for an IoT environment are currently located in transportation and logistics, health care, smart environments, personal and social, and much more. A literature review of Atzori, Iera, and Morabito [3] provides an overall overview of these fields of application and focuses in detail on topics of security and privacy concerns.

As previously mentioned, IoT applications sometimes use WSNs to establish a wireless interconnection between devices. Especially, a WSN is used for connecting several nodes together inside a specified environment with advantage of low-power constraints. The usage of WSNs inside an IoT architecture implies to translate the used WSN protocols into an IoT preferred one. This translation is necessary to transmit data from a low-power environment into the big world of the Internet.

2.2 WSN (Wireless Sensor Network)

A WSN consists of a large number of sensor nodes, which are interconnected via a wireless radio protocol. These sensor nodes are densely deployed over a specified area of interest. WSNs are typically used for monitoring physical or environmental conditions. The fields of application are wide spread from home automation, environmental monitoring, over health care up to military applications. In the scope of home automation is one target to make life better and smarter. Another important field of application is health care because in this field WSNs can make it possible to monitor patients in a less invasive way. Most applications have the target to reduce costs, in relation to adopt dynamical system artefacts to their present requirements. For example, street lights are only switched on and off at the position where a passenger currently walks. The amount of fields of application is very high, due to the fact that a wireless system can be easily deployed at many various places with low operational costs.

Figure 2.1 illustrates a typical structure of a WSN with several sensor nodes, routers and finally a gateway. The topology of some architectures requires the usage of routers, which extends the communication range between nodes. The gateway works as translation unit between the WSN and other networks.

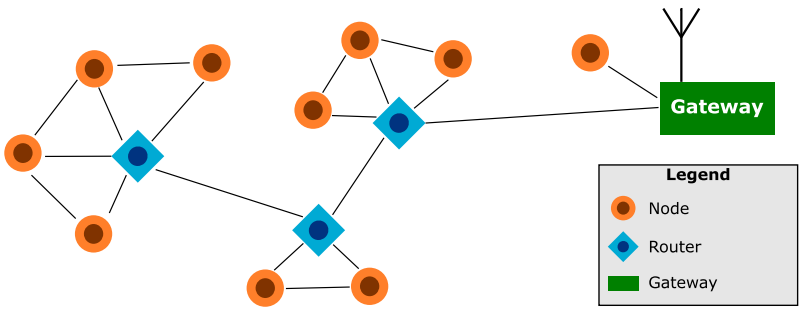
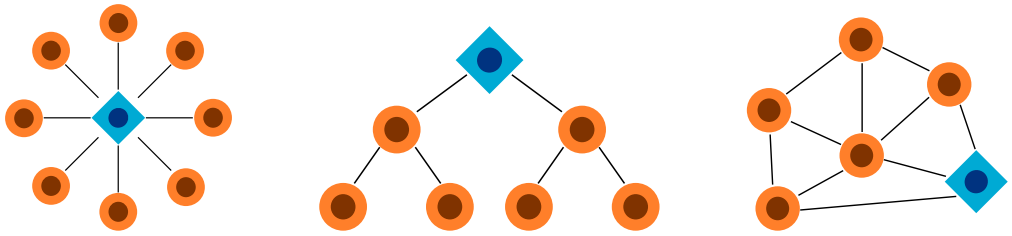


Figure 2.1: Typical structure of a WSN with several sensor nodes, routers, and a gateway.

2.2.1 Topologies

A common WSN architecture consists of several sensor nodes and one central base station. The base station is frequently named gateway and receives all data from the nodes in the established network. Additionally, the gateway works as linking unit between computers and the sensor world. The communication topology can be organized in three main types: Star, Tree, and Mesh [4]. The star topology has one central gateway, which communicates with all sensor nodes. In a cluster tree network each node communicates to his parent node, until the message reaches the gateway. The most flexible and reliable network topology is the mesh network because in this topology each sensor node can communicate with each other for passing data through the network. Figure 2.2 depicts the three different network topologies.



(a) Star **(b)** Cluster Tree **(c)** Mesh

Figure 2.2: This combination of figures demonstrates the three different network topologies for a common WSN. The round, orange objects visualizes nodes and the blue objects are gateways.

2.2.2 Sensor/Actuator Node

A sensor/actuator node contains several technical components in order to perform in a WSN. Such a node can operate as a measurement unit or also as a controlling unit. Figure 2.3 depicts the most common components of a sensor/actuator node. These components are a power management unit, a microcontroller, interfaces to sensors/actuators, and a radio unit. The sensor interface measures or actuates in the physical environment, and typically measurement units are humidity, temperature, pressure, and sound. A common use case for a sensor node is to operate during a long time with only one battery charge. Due to this specification of low-power, the power management unit has to provide several energy-saving possibilities. In order to reach a long operation, each component of the node has to be improved to consume as little energy as needed. Another specialization of WSN nodes are the tiny building sizes of the complete architecture. The power management unit can be designed in a way to operate only with a normal battery or by using energy harvesting methods, which can extend dramatically the operating time [5].

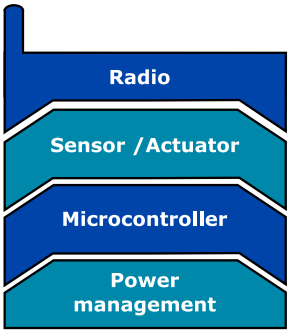


Figure 2.3: Main components of a typical sensor/actuator node in a WSN environment.

2.2.3 Security Requirements and Common Attacks

Security, in a WSN, is important to protect transmitted data over the wireless communication channel. The reason for this procedure is that in some applications, like in health care, transmitted sensitive information of the monitored patient and only authorized individuals should have access to these data. Adding security functions to the system introduces new cost factors like increasing execution time and memory storage. In connection with these factors, power consumption of the overall architecture increases. The new resulting energy consumption is in opposition to a long battery lifetime. As a result, it requires to make compromises between security integration and sensor nodes specifications (energy restrictions, processing power, ...) [6].

Security aware protocols should be designed to take care of different security attributes. These security attributes are among other things: confidentiality, authenticity, integrity, availability, non-repudiation, freshness, forward secrecy, and backward secrecy.

Confidentiality

Confidentiality is one of the fundamental security services, and keeps the privacy of transmitted data packages among sensor nodes. A common way to support confidentiality is to encrypt sensitive data before transmitting them to other nodes. Subsequently, the receiving node has to decrypt the received message in order to get the plaintext.

Authenticity

Every node should be able to verify if the received message was send by a trusted sender or not. The process of authentication usually involves several proofs of identity. This proof could be established by knowing a secret information like a password.

Integrity

The integrity of security assures that the transmitted data was not manipulated during transportation. For the detection of manipulated data packages is created a Cyclic Redundancy Check (CRC) value, which is additionally encrypted with a cryptographic key.

After the creation, the data will be transmitted with the encrypted CRC value. The receiver decrypts the attached encrypted CRC value and compares it with the calculated CRC value. If these two values are not equal, the transmission was manipulated.

Availability

A WSN should be available all the time under normal operation conditions. An attacker could try to change this operation by compromising the network in order to reduce the performance. The worst case of this attack would be a denial of service. This situation occurs when the attacker interferes the radio or disturbs the network protocol.

Security Attacks

A normal WSN provides several channels, which can be used to attack the complete network infrastructure. A WSN is more vulnerable to attacks than other architectures because of the wireless communication unit and their restricted usage of resources. In general, attacks can be classified as active or passive attacks [7]. Passive attacks are not as harmful as active ones because they are only monitoring and listening to the communication channel. Active attacks are more dangerous to a WSN because the intention of an attacker is also to modify messages or to harm the entire transportation.

Some of common active attacks are [8]:

a) **Denial of Service (DoS) attack**
A DoS attack is caused by disturbing the radio communication at the specified frequency. During this attack a normal communication between the affected nodes is not possible. One defense solution would be to use a spread-spectrum communication [9]. This solution is usually restricted by the use of only simple radio units, due to the limited resources of low-power and small construction size.

b) **Battery Drainage**
This attack attempts to keep the sensor nodes awake in order to waste their energy storage. After some time of the attack, the sensor nodes exhaust their entire energy reserves and the devices stop to work. This kind of attack belongs to the DoS group and is called Denial of Sleep [10].

c) **Collision Attack**
Another way to disturb the communication of nodes is to perform signal collisions in the network. This means that the attacker listens to the entire communication and interferes the signal with it's own. Only few changes, some bits of the message, in the communication are enough to produce errors or completely damage the entire message. This attack is better than a DoS attack because this attack affects the communication immediately and not when the battery storages are empty. In addition, this attack is hard to detect because to recognize the malicious device is a complex mechanism. A countermeasure for this attack would be to use error-correcting codes but these procedures add extra overhead to the transmission. Consequently, the general power consumption increases [9].

d) **Sinkhole Attack**
The attacker of a sinkhole attack attempts to attract all the traffic from the nearby nodes in order to redirect the transmitted messages. The main idea of this attack is that the compromised node listens for the request of route information from other nodes and it responds to the request with false route information [11]. The outcome of this attack is that the communication inside the WSN is not working properly any more. Figure 2.4 shows the impact of this attack, which mainly exploits the routing algorithm to harm the communication between the nodes.

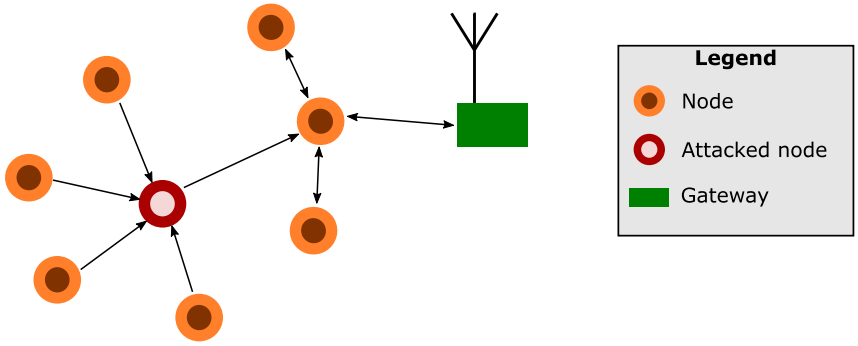


Figure 2.4: This Figure shows the impact of a successfully executed sinkhole attack with several nodes. The attacked node attracts all communications to itself. Illustration based on [8].

e) **Sybil Attack**
In a Sybil attack the attacker owns multiple identities. These identities can be stolen from other nodes or the attacker fabricates new identities [12]. A malicious node with multiple identities is then called Sybil node. The consequence of this attack is to harm significantly the routing protocol. Besides the changed routing information, the routing tables are saturated in the nodes with incorrect information. The result of such an attack is depicted in Figure 2.5.

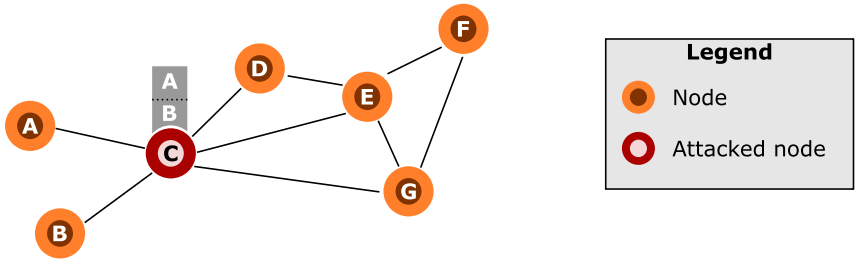


Figure 2.5: This Figure demonstrates the result of a Sybil attack in a WSN. Node C represents the attacked node and has stolen the identities of the nodes A and B. Illustration based on [8].

2.3 Established Protocols

For an IoT or WSN environment, it is essential to have a proper communication protocol, which meets the required properties. These properties define in general the communication topology, used addressing method, energy consumption, and security features. Due to the large amount of various characteristics exists different protocol standards on the market. The next section focuses on some commonly used protocol standards: ZigBee, Z-Wave, IPv6 over Low Power Wireless Personal Area Network (6LoWPAN), and Bluetooth Low Energy (BLE).

2.3.1 ZigBee

The first release of ZigBee was in the year 2004. The specification was developed by the ZigBee-Alliance [13], which is an association of approximately 400 companies. There are already several released versions of ZigBee. ZigBee is mostly used in the application field of home entertainment and home automation. In addition, the standard is also popular in the usage of industrial controlling applications, smoke detectors, in WSN solutions, and much more.

List of available version:

- ZigBee2004 specification [14]
- ZigBee2006 specification (Cluster Library) [15]
- ZigBee2007 specification (ZigBee PRO) [16]

Properties

ZigBee is developed to contribute in a WSN with the characteristic to be a low-cost, low-power, and a wireless network standard. ZigBee modules are typically a full integrated chip with radio unit and microcontroller. ZigBee can be used on the Industrial, Scientific and Medical (ISM) frequency band on the frequencies of 2.4 GHz and on the country specified official ISM band. In China is the authorized frequency at 784 MHz, in Europe at 868 MHz, and in the USA at 915 MHz.

Inside the network of ZigBee can be used three different network topologies for enabling the network infrastructure. These topologies are: star, tree, and mesh networking. Inside the network topologies is a coordinator device, which is responsible for controlling and managing the complete network infrastructure. The final location of the coordinator depends on the used topology. Each device can be chosen inside the network to be responsible for these tasks. The complete network stack is built on the top of the physical layer and the Media Access Control (MAC) of IEEE 802.15.4 [17]. Figure 2.6 illustrates the complete network stack construction of the ZigBee standard.

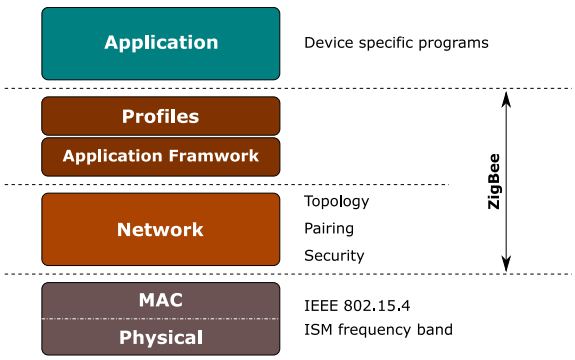


Figure 2.6: Network architecture of the communication stack of ZigBee.

The protocol standard is additionally grouped in several profiles. Each profile is defined for a special field of application. Some defined profiles are for example Home Automation, Smart Energy, Health Care, Light Link, and etc. Profile of this list defines exactly the process of usage and the set of available command structures. This standardization in profiles makes it possible to communicate with miscellaneous ZigBee devices of different manufacturers because of the detailed specification of the available command set. Consequently, a light switch of one manufacturer can control the light bulb of a different manufacturer without translating the commands to another protocol.

At each new ZigBee specifications are added continuously more features to the standardization and additionally improves the usability as well as the security.

Security

The security features are based on the security framework of IEEE 802.15.4. Therefore, it provides a secured communication relaying on a symmetric cryptographic algorithms like an Advanced Encryption Standard (AES) cipher with 128-bit.

The key distribution is one of the key procedures to provide an adequate security level. In previous versions of ZigBee, new cryptographic keys encrypted with the global master key or completely in plaintext were transmitted. This master key is stored in the hardware module and protected from being changed because all devices inside a specific profile require the same initial key. After some time, the master key was figured out by someone and through the knowledge of the master key the security has been compromised.

The countermeasure for this procedure was the introduction of a new security architecture for new versions of ZigBee. In the new security standard is requested a trusted center, which coordinates the other nodes and guides the key exchange in a secured way. The complete ZigBee architecture is split into several levels, which is secured by various cryptographic keys: Master key, Link key, and network key. Further, the network is categorized into two types, namely in a standard application and in a high security application. These two categorizations are protected on the same way but are independently handled. A big change in the process of pairing is the introduction of authentication of new ZigBee nodes before they can be integrated into the existing network infrastructure. The overall security architecture is described in [18].

Recent Attacks

Recently some attacks are known to harm ZigBee, in order to disturb the communication or send manipulated data messages. One attack is a DoS attack, where the attacker sends data packets to the desired nodes periodically [19]. These data packets set the frame counter value to the maximum number and the payload is randomly chosen, which corresponds to an encrypted message. The node tries to encrypt the received message to a meaningless plaintext and sets the internal frame counter reference value to the maximum number. The result by accepting this message is that now all further received messages are rejected from other nodes with a legitimate frame counter value because the included frame counter will be lower than the internally reference value. Consequently, the network infrastructure is disturbed and the battery level is as well drained, due to the amount of retries in message transmission. This attack is only possible when the node does not verify the encrypted payload with a Message Authentication Code (MAC) value.

In the scientific work two possible attacks on ZigBee devices are presented [19]. The first attack is called ZigBee End-Device Sabotage attack. The main tasks of this attack are to impersonate the ZigBee router or coordinator and send continuous broadcast messages to the nodes. Due to the broadcast messages, the nodes will remain in the active state and consume more energy than in sleeping mode. The consequence of this attack is that the nodes run out of power and consequently produce power failures. In worst case, the node provides unauthorized openings.

The second attack is discussed in [19] and performs sniffing attacks on the network key. In older versions of ZigBee it is possible to transmit the network key unencrypted over-the-air to the devices by using the standard security level. For the execution of this attack, only a ZigBee enabled transceiver device is required, which can be listen to the entire network communication. With the support of special software, it is able to visualize the sniffed network stream and to display the messages in a human readable way. In the variety of received messages, it is possible to extract the unencrypted network key.

2.3.2 Z-Wave

Z-Wave [20] is another wireless protocol for WSN applications. Z-Wave was evolved by Zensys and was confirmed by the Z-Wave Alliance [21]. This protocol standard is designed for home automation and commercial environments. For the application field of home automation, there are already a lot of different products available on the market which enable the functionality of controlling lights, air conditioner, oven, television, home security, and much more.

Properties

The Z Wave network is comprised of a mesh network architecture. The mesh network enables each node to communicate with other nodes on a shortest path. Before a new Z-Wave device can operate in the network it has to be paired with a controller by pressing a sequence of buttons.

Each established Z-Wave network has an own home IDentification (ID) and all included devices inside the network are additionally identified by a node ID. The home ID consists

of 4 bytes and the node ID consists of 1 byte. Some node IDs are reserved for internal communication and special functions. Consequently, one home ID can operate a maximum of 232 Z-Wave devices. Nodes with different home IDs are not able to communicate with each other because different home IDs are isolated from each other.

Z-Wave devices operate in the unlicensed frequency bands of the ISM standardization. The exact used frequency depends on the published country because in each country or continent are defined different carrier frequencies. Z-Wave avoids particularly the crowded 2.4 GHz carrier frequency and is consequently more robust and reliable than a lot of other communication participants with different technologies like WiFi and BLE.

A list of typical carrier frequencies are listed below:

- **Europe:** 868.42 / 869.85 MHz
- **United States:** 908.4 / 916 MHz
- **China:** 868.4 MHz

The communication protocol of Z-Wave is constructed on the top of the ITU-T G.9959 standardization [22]. The detailed network architecture is illustrated in Figure 2.7.

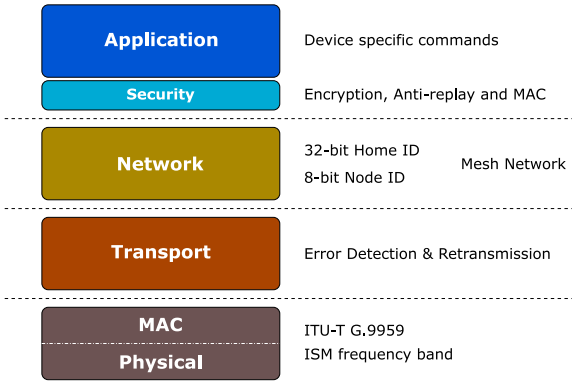


Figure 2.7: Network architecture of the communication stack of Z-Wave.

Security

The security model of the Z-Wave protocol provides several security levels. The first versions of the protocol supported the security enhancements of enabling AES encryption with 128 bits. In the current point of view, the key agreement process is obsolete due to the reason that there already exists some common known security attacks, which are described in the subsection “Attacks” of Section 2.3.2.

Due to the founded security flaws, Z-Wave Alliance developed a new standardization and introduces a new S2 security solution [23]. The Z-Wave S2 Security introduces three different security levels: S2 Access Control, S2 Authenticated, and S2 Unauthenticated. Each security level has a unique network key. The highest security level exists in the S2 Access

Control. For exchanging network keys, a temporary Elliptic Curve Diffie-Hellman (ECDH) key is used for processing the key exchange. The S2 security combines authentication and nonce scrambling to ensure a high security level. Another security feature in the pairing process of devices is authentication, which means that each device has to be verified by its local existence in the area of the network. In detail, a joining node needs to be verified by a Device-Specific Key (DSK) string of decimal digits. This DSK can be read visually or scanned as a Quick Response (QR) code. The DSK is the first part of the ECDH public key of the joining node.

Recent Attacks

In the Z-Wave protocol, there exist attacks which exploit vulnerabilities of the protocol standard and faults in firmware implementations. One approach is to use the Radio Frequency (RF) channel to inject packages to an existing Z-Wave network. Fouladi and Ghanoun [24] demonstrated one of the first public attacks on the Z-Wave protocol. They attacked a door lock with Z-Wave protocol and discovered an implementation error that allows one to reset the established network key. In the first stage, the packages have to be captured to receive the Home ID of the controller and the desired Node ID. After this, the network key can be reset to any known key. The result of this attack is that the attacker is able to take over the door lock. Consequently, the attacker can open or close the door lock whenever they want. This vulnerability is not directly related to the protocol because the error was in the additional developed firmware of the Wave door lock. Nevertheless, this attack illustrates that a wrong usage of the required key exchange can lead to a big security gap.

Another vulnerability was discovered with the Scapy-Radio project [25]. The Scapy-Radio project combines Scapy with the gnuRadio software on a Software-Defined Radio (SDR). This software enables the possibility to capture the wireless network traffic and to replay packets to the network. In the first stage of this project, the traffic of a Z-Wave enabled alarm system is captured. During the process of analyzing the transmitted packets, the ON and OFF command is not changed in the Z-Wave network. With this information, it is possible to intercept the connection to an alarm device and subsequently inject an OFF command to the alarm system. The consequence of this attack is that the alarm system can be disabled by injecting previously captured data packets. The attack was possible due to the usage of a low security level.

2.3.3 6LoWPAN

6LoWPAN is a developed standard from the Internet Engineering Task Force (IETF). The fundamental idea of this standard is to bring the Internet Protocol Version 6 (IPv6) to small and low-power sensor nodes, which are designed for WSNs, and to be able operate in a large IoT environment. One benefit of this protocol standard is the process of addressing connected devices because through the ported version of IPv6 it is possible to connect a WSN to the standard IPv6 network inside an existing infrastructure [26]. In general, this procedure simplifies the connectivity model of the the entire IoT infrastructure. In some use cases, border routers are required, which are used for translating between diverse physical communications layer without changing the process of addressing. Furthermore,

6LoWPAN is an adapted standard of IPv6, in order to be able to transmit packets over the IEEE 802.15.4 physical layer standard. In recent literature reviews are also attempt to use other physical layers like BLE for the 6LoWPAN communication stack. Another advantage of 6LoWPAN protocol is that the usage of the standardization is royalty-free.

Properties

In 6LoWPAN, two types of network topologies are supported: star, and mesh topology. Inside of a star topology, all nodes require a communication channel to the coordinator in order to forward messages to the final receiver. The advantage in the mesh topology is that each node can send the messages directly to the other nodes by using low-power routing protocols.

Figure 2.8 depicts the entire architecture of the communication stack of 6LoWPAN. As the Figures illustrates, the application layer is not directly specified. This option leads to the fact that for each node several applications can be implemented, which are tailored to their requirements. The typical sockets provided in the transportation layer are as follows: User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and Internet Control Message Protocol (ICMP). In IoT applications UDP sockets are typically used, due to performance reasons because UDP sockets have less overhead in comparison to TCP. These layers correspond with the Open Systems Interconnection Model (OSI-Model) [27].

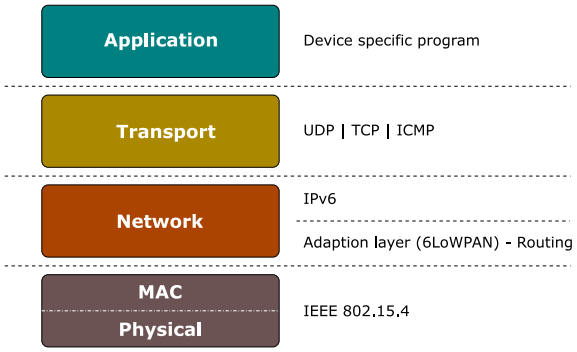


Figure 2.8: Network architecture of the communication stack of 6LoWPAN.

In the network layer, the process of routing messages to the available nodes is also defined. Two approaches are specified: IPv6 neighbor discovery, and Routing Protocol for Low-Power and Lossy Networks (RPL). The used routing protocol is not completely specified for all networks, therefore to be able to connect into the existing IoT environment with 6LoWPAN, it is required to know the used routing protocol.

In order to be able to handle the big IPv6 headers in a low-power environment, several features in the header construction are implemented. These headers are Dispatch Header, Mesh Header, Fragmentation Header, and a header compression functionality. For each activated functionality like mesh and fragmentation, there is a reduction in the maximum transportable payload size by the additional size for the headers.

Security

The communication has to be secured inside a 6LoWPAN network in order to exchange data in a trusted way. The network standardization provides several techniques to ensure a secured environment. These implemented techniques have the task to provide at least the authentication of the sender, confidentiality of data, integrity of the frame, and the network availability [28]. These properties can be archived by enhancing the various layers of the network stack with security features. Convenient layers are the link layer, network layer, or the application layer.

The first entry point of integrating security is to introduce a link layer encryption. IEEE 802.15.4 has already implemented security features, which provides data encryption and authentication. For the encryption process several security architectures are provisioned. One of them is AES in Cipher Block Chaining (CBC) mode including MAC authentication with a key length of 128 bits. The procedure of exchanging cryptographic keys between participants is not specified in the specification.

The network layer provides the possibility to enable the IPsec [29] protocol. The IPsec protocol secures each Internet Protocol (IP) packet by encryption and authentication. The advantage of IPsec is the provided functionality of establishing an end-to-end security channel between various devices. For 6LoWPAN a compressed IPsec standardization is designed, which is stated in [30]. Another benefit of the enhanced security in the network layer is that the communication of each application is secured automatically. IPsec uses two different modes for operation: transport and tunnel mode. In the transport mode, are only encrypted and authenticated the IP packet [28]. In this mode the routing protocol is not protected. In tunnel mode the entire IP packet is encrypted and authenticated, which encapsulate the original IP packet into a new one.

In the application layer, several approaches are available to provide security enhancements in the connection to other participants inside the network.

Recent Attacks

A pure 6LoWPAN protocol stack is vulnerable against several attacks. The literature review of attacks in 6LoWPAN networks showed that attacks are possible in the process of RPL, packet fragmentation, denial-of-services, and much more.

The work of Hummen, Hiller, Wirtz, et al. [31] describe an attack on the fragmentation process in the 6LoWPAN network. The attack uses the fragmentation in combination of routing mechanism to deny the correct processing of legitimate fragmented packets. The attacker has several scenarios on how to exploit the usability of fragmented data packets. In the first one, a transmitted fragment is duplicated inside the network by sniffing and retransmitting the packet by the attacker. The receiver node receives both data fragments: the original, and the duplicated one. At this moment it is not clear for the receiver node to decide, which fragment is the correct one. Consequently, the entire packet will be discarded. The second variant is to transmit several new fragmentation packets to reserve the buffer in the receiver nodes; therefore, a normal data packet will not fit into the memory and the packet will also be discarded. A countermeasure for this attack is a secured 6LoWPAN fragmentation header structure.

Another attack is cited in survey [32], which focuses on the usage of RPL. Without enabling security features inside the 6LoWPAN network, it is easy to attack the infrastructure on several levels. The stated researching work describes the different attack vectors in the network when a RPL is used for routing data packets.

2.3.4 BLE (Bluetooth Low Energy)

BLE is classified to the group of wireless personal area network technology and is a sub technology of Bluetooth. This technology is especially designed for low-power applications like fitness trackers, home automation, security, health care, and much more. BLE is also known under the market name of Bluetooth Smart. The difference between the low-power BLE and the normal Bluetooth is the reduced power consumption and operational costs, while the communication range is nearly the same.

Properties

BLE is a communication protocol for low-power devices which allows to communicate with standard devices like smartphones and similar devices. Currently, this protocol only supports a star network topology. In the center of the communication infrastructure is a master node, which manages the surrounded slave nodes. Research projects have presented first approaches to introduce a mesh topology to BLE with the name BLEmesh [33].

BLE also operates in the ISM frequency band in the frequency range of 2.4 GHz to 2.4835 GHz, such as the standard Bluetooth configuration. The differences between these two specifications are that BLE uses a smaller set of channels, and have a reduced data rate.

Security

BLE in the version 4.2 [34] provides a new security manager for secured data transmissions. The security manager module specifies the process of pairing, key distribution and the used security.

In the work of Kwon, Kim, Noh, et al. [35] the detailed process of pairing between two devices is described. The process is split into several steps, which are required to succeed the pairing. In the first step, the authentication requirements and input/output capabilities of the devices are exchanged, in order to determine the method of pairing. Three different methods are available. “Just Works” is used when no input capability is possible for example headsets. Another method would be a “Passkey entry”, which uses the keyboard to enter six digits. The last method is the “Out of Band” method, where the devices use an extra interface for transporting the keys to each other. In the final step of the pairing process, the participating devices create new cryptographic keys under the instructions of one of the above named methods. After these steps, the communication between the two devices is secured.

The various versions of BLE differ in the selected input values for the pairing processes. In version 4.2 a new secure connection pairing process is introduced, which extends the previous processes with additional cryptographic functionalities.

BLE in the versions 4.0, 4.1, and 4.2 (downward compatibility mode) use following workflow:

- **Just Works**
  The used temporary key is set to zero. Due to this procedure, it is easy for an attacker to brute force the new processed key. Consequently, it is possible to eavesdrop the connection and has no protection against man-in-the-middle attacks.

- **Passkey**
  In this method a six digit number is created on one device of the two participants. After the creation of the six digit number, the user has to read the digits from one device and has to enter it on the other one.

- **Out of Band**
  The temporary key is exchanged by using different wireless communication technologies like NFC or similar. An advantage in this process is, that a large temporary key can be transmitted in one step and the complete key transportation is decoupled from the common communication channel.

In version 4.2 a new workflow is introduced, which is only compatible with devices of the same Bluetooth version:

- **Just Works**
  The renewal in this version for this method is that public key cryptography is used to exchange the cryptographic keys. ECDH is used, which ensures a secured transmission of the processed keys.

- **Passkey**
  The entering process of six digits stays the same. The enhancement in this version is the usage of public keys of Elliptic Curve Cryptography (ECC) to authenticate the connection.

- **Out of Band**
  In this process, all required information is exchanged for the pairing over an external wireless communication channel. These are the public keys of ECC, processed nonces, and confirmation values.

Recent Attacks

Up to the BLE version 4.2 exists a vulnerability in the pairing process in creating new keys. These vulnerabilities are stated in work [35]. In the pairing processes are used input values, which are transmitted through a temporary encrypted packet. The transmitted length of the unknown temporary key is too short to provide a high security level. Consequently, the transmitted cryptographic key can be predicted by an attacker through a simple brute force attack.

Another vulnerability has been discovered in the “Just Works” method. In this method, the predefined temporary key of zero is always used due to the restrictions of having no input possibilities. Consequently, the generated cryptographic key is exchanged in plaintext.

Furthermore, in the "Passkey Entry" method, it is possible to brute force the six digits in a short period of time. As countermeasure of this brute force attack, the number of digits have to be increased.

2.4 Key Management

Good key management is necessary for a WSN to operate in a trustful environment. Therefore, all key management schemes should fulfill the typical security requirements. These security requirements are: confidentiality, authentication, freshness, integrity and non-repudiation. Key management can be separated into two main groups: static, and dynamic key management schemes. Dynamic key management schemes offer more advantages in the aspect of security for WSNs. One reason for this argument is that the network structure of a complete WSN can change dynamically over time, relating to connecting new nodes or losing them. Consequently, the dynamic key management will be handled in more detail in this section.

The main task of a dynamic key management procedure is to provide and manage cryptographic keys in a secured way. Additionally, it is important to provide methods to revoke the permission to operate in the current network from malicious nodes. Special systems like intruder detection systems can detect compromised sensor nodes. If a compromised sensor node is detected, then the key management should perform a new key distribution process to all other nodes. A dynamic key management should achieve following properties [36]:

a) **Forward and Backward Secrecy**
Forward secrecy is to protect the sensor node by using an old key to decrypt new messages with it. The same is valid for the backward secrecy. With the new key it should not be possible to decrypt the old data packages. Both secrecies are used to prevent node capture attacks.

b) **Node Revocation**
Node revocation is a procedure to remove compromised sensor nodes promptly from the current network structure. If a malicious node is detected in the network, then the cryptographic keys should be updated at all nodes excluding the malicious one. This procedure prevents the network from harming the network communication by modified data packet and injected incorrect messages.

c) **Collusion Resistance**
The collusion resistance describes the type of method when an attacker tries to compromise a portion of sensor nodes to receive all system keys. With these system keys it would be possible to capture the entire network and eavesdrop on the communication.

d) **Resilience**
Resilience is one indicator of the resistance against node capturing. In a WSN, low resilience means that the captured data of one node leads to compromise the complete network infrastructure. The opposite is a high resilience, where an attack only affects

one single node and not the entire network. Generally, an attacker captures a lot of data packages and tries to recover secret information like cryptographic keys.

Dynamic key management systems can mainly be separated into two groups. One group is specialized for distributed key management and the other one for a centralized one.

2.4.1 Distributed Key Management Schemes

A distributed key management has no central key controller, like a base station or a third party, for establishing a rekeying process of nodes. Consequently, the rekeying process is handled by multiple key controllers. This approach avoids single point of failure and satisfies better network scalability.

EBS-based Key Management

Exclusive Basis System (EBS) is a group key management with a combinatorial formulation for WSN [37]. In this scheme, a specified number of keys out of a pool are assigned to a single node. The process of rekeying can be triggered periodically when the network structure has been changed, or a sensor node is captured.

Some EBS based key managements are listed and described below [36]:

a) **SHELL**
A Scalable, Hierarchical, Efficient, Location-aware and Lightweight (SHELL) network consists of one command node, cluster heads, gateways and sensor nodes. The command node is assumed to have enough resources and cannot be compromised. The sensor nodes are grouped into different clusters. A refreshing of keys can be performed inside one cluster or among the connected cluster heads. In this key management scheme, a collusion prevention heuristic key assignment is proposed, in order to increase the number of colluding nodes for eavesdropping the whole network. Therefore, the nodes with a short physical distance are assigned with keys with a lower Hamming distance than those which are further away. The disadvantages of SHELL are that it is a highly complex heterogeneous node operation, where multiple types of keys are used, and the physical location of nodes is known.

b) **LOCK**
Localized Combinatorial Keying (LOCK) scheme consists of three different hierarchical levels. At the top of the hierarchy is the base station. The base station is followed by several cluster leader nodes. Behind the cluster leader nodes are the regular sensor nodes. During a rekeying process no location information can be used for generating a new key. This scheme provides different rekeying scenarios for captured sensor nodes and cluster leaders. The advantage of this scheme is, that the nodes are locally rekeyed, hence its reduces the time delay of generating new keys and reduces the energy consumption.

Polynomial Secret Sharing Based

In this scheme all nodes are randomly assigned to groups. Each group shares a unique key between all node members. In the basic scheme one-hop neighbors are required to protect their group polynomials collaboratively. If one sensor node and a certain amount of numbers of its one-hop neighbors are compromised then the group key polynomial could be revealed.

a) **Cluster-based**
A cluster-based group key management system was proposed in a work from Zhang et al. [38]. This scheme generates and distributes a group key to the nodes within a cluster, instead of the collaboration of neighboring nodes to acquire a group key. The complete system is separated into several parts. The first part is to initial a sink node, which decides the total number of groups. For each group, a unique $2t$ degree bivariate polynomial $g(x, y)$ in constructed over a prime finite field. Furthermore, each node u gains its personal secret $g(u, y)$. For the channel selection and cluster formation an algorithm is used which uses a one-way hash function with the identifier of its member nodes for each channel to construct hierarchical keys. The current group key can only be derived with a broadcast request message to all included nodes of the cluster. All included nodes will respond with the personal secret $g(u, y)$, where y is the group key. Afterwards, the channel unicast the current group key in a secured way to its members by using the hierarchical key. This key management system also includes mechanism to exclude compromised nodes from the cluster.

Deterministic Sequence-Number-Based

In deterministic sequence-number-based key management systems broadcasts, each node choses randomly a number. This random number is used for establishing a pairwise key between their neighbors. This basic structure exists in various schemes [39], which is described below in more detail.

a) **LEAP Scheme**
Localized Encryption and Authentication Protocol (LEAP) is a key management scheme, which provides multiple keying mechanism for providing confidentiality and authentication in networks for sensors [40]. The base station in a sensor network generates an initial key in a pre-deployment phase and inserts this key in all sensor nodes. After the initial phase, each node can derive its master key with the identifier and the pre-deployed key. In a rekeying process inside the group, the cluster head updates pairwise the keys with the nodes. After all pairwise keys are updated, the channel head randomly chooses a new group key and encrypts it with the pairwise keys and send it to the nodes. This rekeying process has its flaws; the pairwise key always uses the same algorithm and the initial key. If an attacker captures a sensor node, then the attacker is able to reveal the initial key and consequently all pairwise keys can be easily computed.

b) **OTMK Scheme**
Opaque Transitory Master Key (OTMK) [41] is a key management system, which improves the LEAP scheme. In this scheme the nodes are pre-assigned with a master

key. If a node wants to establish a pairwise key to a neighbor node, then it has to broadcast at first an encrypted join message by using the master key and a random number. If both nodes receive the join message, then they use their ID and a random number to compute their pairwise key. The rekeying process follows the LEAP protocol. Consequently, if an intruder knows the pre-assigned master key it is also possible to eavesdrop the complete network.

c) **EDDK Scheme**
The proposed scheme Energy-efficient Distributed Deterministic Keymanagement (EDDK) [42] should handle the resource exhausting attacks and DoS attacks. In EDDK each node is pre-assigned with a pseudo-random function, an initial key, and a local group key. Each node creates a table of neighbors with the cryptographic keys and the random numbers. The process of joining a new node to the network is closely related to OTMK and the process of rekeying to LEAP scheme. EDDK is developed to prevent reply attacks, Sybil attacks and node replication. The disadvantage of this scheme is that it is not suitable for large WSNs, due to the large memory consumption.

2.4.2 Centralized Key Management Schemes

The centralized key management scheme uses a single central key controller, which is completely responsible for the key management. This key management can further be separated according their network structures: A flat network, hierarchical, and heterogeneous.

Flat Network Based

In a flat network, all sensor nodes have the same functionality. The nodes are directly connected with a base station and share a secret key.

a) **KeyRev**
A flat network based key management is the KeyRev [43], which is an efficient scheme for removing compromised sensor nodes from WSNs. This scheme assumes that all nodes can communicate directly with the base station. Each node has to maintain several keys. These keys are a pairwise key, path key, encryption key, and a key for MAC. The specialization of this scheme is that the lifetime of a WSN is divided into sessions. The session key is provided by the base station. If a node wants to join a network with KeyRev, then the node has to be pre-loaded with the different keys and the personal secret. KeyRev security efficiency also depends on the accuracy of the compromised nodes detection and the lifetime of a session.

b) **EEKM**
Energy-Efficient Key Management (EEKM) [44] is designed for large scalable WSNs. In this scheme it is important that the base station can directly communicate with the nodes through broadcast messages. All sensor nodes are separated into different virtual and regional groups. Each node is pre-assigned with an initial master key. This master key is used to generate a group key and pairwise keys. The rekeying process is regional-group orientated.

Hierarchical Network Based

The hierarchical network based key management is constructed in a tree structure. One base station controls and manages the key exchanges and the process of rekeying.

a) **Spanning Tree Key Management (STKM)**
The nodes in a STKM are connected in a tree structure [45]. Each sensor node contains three different keys. Two keys are used for encryption and decryption of messages between the node and the base station. The third key is shared with all nodes and is renewed periodically by the base station. The spanning tree is constructed by broadcasting Hello messages to the nodes. The first Hello message is initiated by the base station. If all reachable nodes have joined the network, then these added nodes can broadcast Hello message. This procedure is repeated until all nodes have successfully joined the network.

b) **Location-Aware and Secret Share Based (LASSB)**
Another dynamic key management is LASSB for grid based WSNs [46]. In this scheme, the original cluster head selects two or more gateway nodes which have a larger energy unit and enough memory for storing keys of the nodes from the closest physical location. The basic idea is to transfer the important keys to different locations in order to be protected towards a base station attack, which would reveal many stored keys.

Heterogeneous Network Based

A heterogeneous network based sensor network consists of a large number of stationary or moveable sensor nodes, and a central base station. The base station manages the network and collects data from the distributed nodes. Sensor nodes can be separated into two groups: nodes with high processing capabilities, also known as cluster heads, and nodes with a low processing capabilities known as cluster members.

a) **Genetic Algorithm Based Key Management**
The Genetic Algorithm based key management scheme [47] uses, as the name already suggests, genetic algorithms to design appropriate functions for the rekeying process. The complete network consists of header nodes, sensor nodes and one sink node. The sink node is responsible for the process of generating appropriate key generation function and to transmit them to the header and sensor nodes. These functions are sets of code slices. Each possible key function, which could be used for generation, is encoded as chromosome.

b) **CL-EKM Protocol**
The Certificateless-Effective Key Management (CL-EKM) [48] supports the establishment of four types of keys. These keys are a certificateless public/private key pair, an individual key, a pairwise key, and a cluster key. The certificateless public/private key pair is used in the setup phase of the node to generate a mutually authenticated pairwise key. The individual key is a unique key between the node and the base station. The pairwise key is used for a secured and authenticated communication with the neighbor nodes. The cluster key is used in the group of some

nodes for encrypting broadcast messages. The complete scheme consists of different processes, which are managing the key generation, key updates, key revocation, and node joining processes.

c) **Public Key Infrastructure (PKI)**
Key management schemes can be based on symmetric and asymmetric cryptographic solutions. In a typically WSN is mostly used only symmetric cryptographic algorithms, due to the limitations of energy consumption and performance reasons. Asymmetric cryptography provides better resistance against node compromise attacks. Nowadays, asymmetric cryptographic methods are also found inside WSNs because of the energy efficient implementations of ECC. Solutions for asymmetric key managements are Tiny Public Key (TinyPK) [49] and Tiny Elliptic Curve Cryptosystem (TinyECC) [50]. These two systems are based on the standard public/private cryptography infrastructure and are designed for tiny WSN nodes.

2.5 Exemplary Smart Home Applications and Devices

This section is focused on some common products for smart home applications. These products are described by their functionality and general properties. If attacks are known for the selected product, then they are briefly mentioned.

2.5.1 Samsung SmartThings

SmartThings is an open platform for IoT applications in home automation. This platform provides a central gateway, or also called hub, and individually installable SmartApps. SmartApps can be installed from an application store by using the smartphone. The basic idea of this product is to connect a lot of different smart devices to one central unit with an easy usability. The SmartThings gateway includes multiple radio communication protocols like ZigBee, Z-Wave, and WiFi. These protocols are used for communicating with the smart devices and is therefore well equipped to connect wide spread of devices.

Researchers analyzed this product on their software framework against security weaknesses [51]. They focused on the framework because this part in the software cannot be updated very quickly, due to the fact that there exists more than 200 external developed SmartApps. In previous work [51], the researchers found two security-critical design flaws: One in the SmartThings capability model and another one in the event subsystem. The SmartApps are over privileged with capabilities, which they do not have been requested. The event subsystem provides information about the events of any devices without having special privileges. The following attacks occur due to these weaknesses:

- Pin code injection at door lock
- Snooping of door lock pin code
- Disabling Vacation Mode
- Enable fake alarm

2.5.2 Philips Hue

Philips Hue light products are designed to revolutionize light control in smart homes. The overall system consists of two separate devices: The light bulbs, and a gateway. The gateway is used as interconnection between the smartphones and the light bulbs. The reason for this procedure is that the light bulbs only uses ZigBee Light Link standard as communication protocol.

Researchers found an attack in this product family to overtake the control of Philips Hue light system [52]. The attack uses an implementation error in the ZigBee Light Link protocol state machine. The total takeover attack is accomplished with a correlation power analysis to encrypt and verify firmware updates. The correlation power analysis revealed the cryptographic key for checking if the new firmware is an original one from Philips or not.

The effectiveness of the attack has been demonstrated with a flying drone. This drone takesover all Philips Hue lamps by updating the firmware through flying past a building. The distance between the attacker and the Phillips Hue light bulbs were at the starting point approximately 350 meters, where the first attacks already succeeded.

2.5.3 Atmel Smart Plug

Atmel developed a demonstrator for a smart home application. This demonstrator is presented in detail with their used components and architecture in the application note AT15735 [53]. The goal of this demonstrator is to demonstrate a showcase of an IoT application by using Atmel components. The smart plug can be controlled with an Android application on a smartphone.

The connectivity between the devices is established with WiFi and uses no low-power communication protocols. WiFi is a common way to connect smart devices among each other, but the power consumption is quite high. Therefore, it would be better to use some low-power communication protocols, including other radio frequency units instead of WiFi.

The transmitted messages between two devices are encrypted and also authenticated, which ensures a high security level. The encryption is based on ECC cryptography. Additionally, in the application note is described the implemented communication protocol between two devices. The designed protocol shows, that the overhead for each message is quite high due to the used cryptography algorithm. The pairing process is established with a smartphone, which is also connected to the same WiFi network. All cryptographic keys are stored on a hardware security controller.

The conclusion of this demonstrator is that the idea and construction is performed in a good way, but the power consumption of the IoT devices should be considered. WiFi is a good established communication channel in a typical home with high bandwidth. A public cryptography requires much more payload data than in comparison to AES in CBC mode. Nonetheless, it has to be analyzed if it can be processed for the same application, with a lower payload size with a comparable security level.

2.5.4 Nest

Nest is another product group for home automation. The product range includes thermostats, smoke detectors, security cameras, and further security system elements. The communication in a Nest Labs network is ensured with WiFi and its own designed protocol Weave. Nest combines products with self-learning algorithms for controlling homes in a smart way.

Additionally, in some processes of pairing the devices inside a Nest environment a BLE for transmitting confidential data between the pairing participants is used. This additional communication channel has turned out to compromise Nest cameras. The BLE module is always on and cannot be turned off, therefore it makes it possible for everyone within the communication range to overwhelm the cameras. During the process of overtaking them, it is possible to shut down the cameras and consequently turn off the protecting feature of video capturing. The security researcher Jason Doyle described the security flaws in [54].

Another security issue was found in the Nest thermostat device. Previous research [55] explains the process of installing malicious firmware on the device through the Universal Serial Bus (USB) connection. Through this process, it is possible to change the entire behavior of the thermostat and can be hazardous for the heating or air conditioner control unit.

2.6 Embedded OS

For the IoT environment, various embedded OSs are developed, which are applicable for miscellaneous applications. The usage of an OS is a big advantage, due to the integrated functionality of network protocols and other modules. Commonly, the OS is developed in a way that it can be configured in a modular concept.

The embedded OS acts in general as a resource manager for a complex platform of various components. A typical platform consists of resources like a processor, network interfaces, memory management, timers, and much more [56]. The fundamental tasks of such an OS are to manage all attached resources and to execute developed applications.

An embedded OS is generally defined by parameters and functionalities, which describes the footprint, scalability, probability, modularity and connectivity. For the usage of an embedded OS in an IoT environment, it is important to receive a tiny footprint after compiling the source code and consequently fits into the small memory storages of IoT nodes. The OS is among other things also responsible for providing energy saving mechanism.

Commonly used OSs for the IoT environment are: Contiki OS [57], Tiny OS [58], and RIOT [59]. Of course there are a lot more on the market, but these ones are open source products, which provides a good starting point to develop own IoT devices.

2.6.1 Tiny OS

Tiny OS is especially designed for WSN to operate on low-power devices [60]. The OS is written in a dialect of the C language, namely in nesC. The main target of the Tiny OS is to be flexible to different hardware platforms and to has a small footprint in code size. The OS consists of a list of components: libraries, network protocols, distributed services, drivers and data acquisition tools.

The architecture of Tiny OS is build up with three interface abstraction layers: commands, tasks, and events. For communications and data exchanges between components, the command and events structure is used. Commands can request the OS to perform several services. After the completion of a requested service, events to signal the completion are used. Tasks describes the intra-component concurrency of implemented services.

Tiny OS provides multi threading support and includes system calls, which can create, destroy, pause, resume, and join threads. The threads are scheduled by a First In First Out (FIFO) algorithm.

2.6.2 Contiki OS

Contiki OS is another embedded OS for tiny nodes constructions. The system architecture consists of several modules: A kernel, various libraries, a program loader, and a set of processes [61]. This OS is organized in processes, therefore each application or service is created and executed as process. Services are used for implementing shared functionalities for more than one application process. A useful property of Contiki is that each process can be separately started and stopped during run-time, therefore it can be used very dynamically.

A process consists of two parts in Contiki: a process control block and a process thread [62]. The first part, the process control block, is stored in the Random-Access Memory (RAM) and contains run-time information about the process itself. The process thread is the code of the process and is stored consequently in Read-Only Memory (ROM). Additionally, the usage of protothreads is implemented inside the processes. This functionality allows processes to wait on incoming events. Protothreads are used for example for the TCP/IP stack for processing the socket connections.

The resulting code size of Contiki OS is depending from the used modules, protocols, and applications. However, the basic system of the OS is larger than in comparison to the Tiny OS, due to the event triggered kernel services and the process scheduler. Tiny OS only uses a FIFO event structure and Contiki uses a FIFO scheduler with the opportunity to poll handlers with priorities.

Contiki OS supports the communication protocol 6LoWPAN and provides several configuration possibility to modify it to the requested behavior.

2.6.3 RIOT

RIOT is also an OS, which is conceptualized for the usage in IoT environments. The OS is designed to fit on small hardware platforms, due to the minimal resource constraints of IoT nodes. RIOT wants to go a step further than Contiki OS and Tiny OS and wants to

close the gap between event driven OS and a modern full-fledged OS, like a native multi threading, hardware abstraction, and dynamic memory management [63].

The central component of RIOT is the micro kernel architecture, which supports multi threading with standard Application Program Interface (API) functionalities. This OS is also real-time capable and uses therefore constant periods for kernel tasks. The constructed scheduler works without the usage of periodic events, thus it is possible to go to idle or sleep mode when no pending tasks are available.

2.7 NFC (Near Field Communication)

NFC is a sub technology of Radio-Frequency IDentification (RFID). In addition, NFC is a communication protocol, which enables two devices to communicate over an electromagnetic field with each other. A new communication can only be performed when both devices are within a close distance. Generally, this technology operates in the ISM frequency band of 13.56 MHz. For the physical communication various types are specified, which are defining the used encoding and signal type.

Nowadays, this technology is used in several applications. Some examples for the fields of application are: contactless banking cards, in social networking (sharing contacts, photos, or other data), ID cards, and so on. Since the technology was integrated in miscellaneous smartphones, the usage of NFC technologies becomes popular. Over the years, the number of NFC-enabled devices have increased, which make it possible to create concepts for new fields of application.

In the field of IoT NFC can also be used to increase the usability of several IoT devices and to design new applications. In a smart home application with NFC support, the technology can be used in several use cases, which are discussed in detail in the work [64]. One possible use case out of this work is to count or identify available products in a fridge. Another one would be to monitor patients and their devices to support their healing process.

2.7.1 Established Standards and Types

NFC is implemented in different variants, which are identified by their used signal type and encoding. In the next paragraphs are described shortly the common used types.

NFC Type-A

Type-A is specified on ISO 14443A and has similar configurations as RFID type-A. In this type a miller encoding technique is used with Amplitude Modulation (AM). The data is transmitted with a rate of approximately 106 Kbps. The signal in type-A changes from 0% to 100% to distinguish the binary data, if it is binary a zero or a one.

NFC Type-B

This standard of NFC is based on the ISO 14443B and is shortly named type-B. Also this standard is similar to the defined standard of RFID type-B. For this transmission, a

Manchester encoding technique is used and an AM modulation of 10%. The detection of binary data is focused on the change of 10%. This means that a binary data zero is 90% amplitude and a one is represented as 100% amplitude.

NFC Type-F

Another NFC specification is from FeliCA, which has defined the signal specification type-F. This specification is similar to the NFC technology, but it is a separated system technology. This technology is faster than the original NFC and is commonly used in Japan. This technology is already wide spread into different applications in Japan.

2.8 Java Card OS

Java Card is a common used technology, which allows to run Java applets in a secured environment. Java Card is implemented in several embedded devices like Subscriber Identity Module (SIM) cards, ATM cards, in security controllers, and so on. The advantage of this technology is the flexibility in the field of application because due to the possibility to program application specific applets.

The main target of this technology is to provide a secured environment for storing sensitive data in an applet specific storage. The security on the Java Card is satisfied by methods of: data encapsulation, applet firewall, cryptographic functions and the applet itself. Data encapsulation means in this point of view that stored data inside the application are executed in an isolated environment and it is not possible to get access to data from another installed applet. The applet firewall manages the execution of several applications inside of one Java Card virtual machine. The Java Card typically supports a variation of cryptographic functionalities for providing a desired security level and to protect the transmission of data with cryptographic functions. The applet itself is a state machine, which receives Application Protocol Data Unit (APDU) commands from outside and has then the possibility to respond on them.

The interface to the Java Card is specified by the standard ISO 7816 [65]. With this interface it is possible to transmit commands to the Java Card environment. These commands are specified as APDU commands.

2.8.1 APDU Commands

In this section, the construction of APDU commands and on their general usage is discussed. The typical construction of an APDU is stated in Table 2.1.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
Header				Body		

Table 2.1: APDU command composition with all available header fields.

The complete APDU command consists of two parts: Header and Body. The header contains several property fields, which are for defining the command type and the appended body. The meaning of each header property is discussed in the following section:

- **CLA** - 1 byte
  Defines the instruction class of the command. This field indicates which class has to be selected inside the Java Card for the command execution.

- **INS** - 1 byte
  This property filed defines the instruction code of the command. For the Java Card, the command that has to be executed inside the selected class will be specified.

- **P1 & P2** - 2 bytes
  P1 and P2 are additional instruction parameters. With these parameters it is possible to transmit two bytes of extra information to the Java Card.

- **Lc** - 0/1/3 bytes
  The property field, Lc, defines the number of bytes of the followed attached command data. The length of this field varies between 0, 1, or 3 bytes. If no data is attached to the body, then the Lc field can be zero and can be removed during transmission. One byte is only used when the length of the command data is between 1 and 255 bytes. If the data is longer than 255 bytes, then the property field has to be 3 bytes long. The first byte must be zero and the other two represents the length of the data field, which can be between 1 and 65535 bytes long. A special case occurs when all three bytes are zero, then the Java Card OS assumes a command data length of 65536 bytes.

- **Data** - n bytes
  The data field contains the data bytes. The length of the contained data is previously defined in the Lc header field.

- **Le** - 0/1/2/3 bytes
  With this property field it is possible to define the expected length of response data. The encoding for this property field is similar to Lc. Zero bytes are used when no response data is expected. One byte is used to define the response length of 1 to 255 bytes. If the transmitted value is zero, then the field represents a response data length of 256 bytes. If the length of Le is 2 bytes, then a response data length can be requested between 1 and 65535 bytes. Furthermore, two bytes of zeros means a maximum length of 65536 bytes. If Lc was not present in the command, then the Le length can also be three bytes long. Therefore, the first byte must be zero and the ongoing two bytes following the same procedure like for two bytes.

The response of an APDU command is constructed with a body and a trailer. The visualization of this packet is described in Table 2.2. The contained data inside the body is the response of the Java Card from the ongoing APDU command. The trailer is a two byte status field, which represents the return value of the executed function. The return value provides the information, if the processed function was successful or stopped due to an error.

Data	SW1 SW2
Body	Trailer

Table 2.2: APDU command response composition.

2.9 Hardware Security Controller

A hardware security controller [1] is a separate hardware chip, which provides especially cryptographic functions and secured storage spaces. Typical applications for a hardware security controller are to provide an API for creating and storing cryptographic keys for different algorithms. Additionally, it can provide a secured storage for confidential data. The complete hardware controller should be protected against side channel attacks and other attacks, which can hamper or reveal internal data.

Hardware secured elements are becoming more and more important in the field of IoT [66]. The reason for this is the increasing request on security enhanced IoT devices. Typical IoT devices inside a WSN are tiny and constructed as low-power and low-resources platforms. These constraints are not suitable for security enhancements because security cost additional power consumption. Therefore, to be able to satisfy the security inside the products, it is suitable to use a dedicated hardware element. This element should move the complexity of operation to hardware supported devices, instead of consuming calculation power of the microcontrollers. This hardware secured element should finally be responsible for providing tested and secured cryptographic algorithms with the possibility to store keys internally. In an ideal way, the hardware secured element creates on request new keys and do not leave the hardware security controller, like a private key part of the public cryptographic methods.

The main security concerns for IoT devices are reasons to provide following security properties [67]:

- **User Identification**
  For devices it is sometimes necessary to verify if the user has the permission to use the system.
- **Tamper Resistance**
  In the worst case when an attacker gets physical access to the device, it should not be possible to reveal any data. Consequently, it has to be protected against side-channel attacks.
- **Secured Execution Environment**
  The runtime environment of the hardware security element should be able to execute applications without providing internal access.
- **Secured Storage**
  Sensitive data should be stored in a secured protected area. This secured storage is protected especially towards side-channel attacks and can be used for storing personal data or cryptographic keys.

[1] Available at www.infineon.com/security

- **Secured Data Communication**
  The meaning of secured data communication is to provide cryptographic functionality to be able to create point-to-point encryptions. This communication channel should be encrypted and authenticated, which ensures confidentiality and integrity of the communication.

- **Identity Management**
  The identity management controls and identifies each component in a system. Additionally, it also provides the function to control the access to resources within the system.

Chapter 3

Design and Concept

This chapter focuses on the design and concept phase of an IoT network for an exemplary smart home application with enhanced security features. The overall system architecture is evaluated on their components. In addition, the used hardware components and used protocol standards are outlined in detail.

3.1 Requirements

This master thesis has the goal to implement a secured environment inside an IoT network, which can further be used for a smart home application. The overall environment with all system components is illustrated in Figure 3.1. This illustration shows the main components of the conceptualized smart home application: A central gateway, several IoT nodes, smartphone, and Internet compatible devices. The gateway is the central point of the overall IoT environment. Therefore, it is responsible to collect all data from the independent distributed IoT nodes. Furthermore, the gateway is also responsible to visualize the received information of the nodes on a website. The IoT nodes are constructed for different fields of application and can be outlined as sensor platform, actuator platform, or a mixture of both configurations. Some typical applications for a smart home environment are for example a smart outlet, a smart switch, a weather station, and so on. For the process of establishing a secured IoT environment, it is required to combine the existing network infrastructure with several cryptographic mechanisms. The design process also has to be focused on the encryption of all communications between the gateway and the nodes, in order to achieve an appropriate security level. One of the security-critical parts is the first pairing of the devices inside the environment. A reason for this importance is that the key exchange of the cryptographic keys must be ensured to be processed in a secured way, in order that nobody can reveal or eavesdrop the keys. The concept of exchanging the keys in a secured way has to be still simple for the user and should not be complicated. If the secured key exchange is as simple to use as the unsecured one, then the users are inclined to use also the secured one.

A prerequisite for this thesis is to be compatible to other demonstrators, which previously have been developed by Infineon and its project partners. This master thesis is one part of a larger project to establish a secured environment inside an existing network. For these reasons and to stay compatible, it results into some constraints for the design process of the

thesis. One constraint is to use the same wireless transceiver system and communication protocols, which are also included in the other demonstrators. This ensures to be able to receive and send data inside the same network infrastructure.

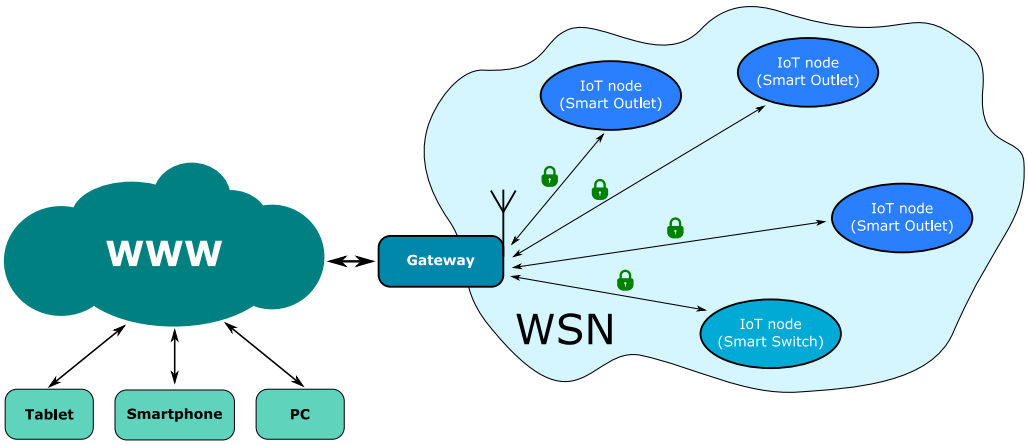


Figure 3.1: This illustration shows the overall IoT environment for a smart home application.

3.1.1 Detailed Requirements Analysis

- **Hardware Components**
  Miscellaneous hardware components have to be designed for the gateway and also for the sensor/actuator nodes. The gateway should provide an interface to the World Wide Web (WWW) and to the WSN. In the design process of the IoT nodes should be considered a low consumption of resources like power and space. Gateway and node-devices have to be equipped with a secured hardware element, which is responsible to store node data and also the cryptographic keys.

- **Contiki - OS**
  The Contiki - OS has to be adapted to fulfill the requirements like implementing functionalities for encryption and decryption. The entire application layer has to be defined and implemented. Some key facts of the application layer are the key management system, webserver functionalities, and management of the communication protocol.

- **Website**
  As previously discussed the website, which is hosted by the gateway, should be responsible to visualize the received information of the sensor/actuator nodes of the distributed WSN. Additionally, the user can also interact and control the IoT nodes with the website.

- **Security Enhancements**
  The new security enhancements for an IoT network should introduce the possibility to configure nodes with the support of a smartphone before they are connected to

the IoT network. In detail should be used NFC technology from the smartphones to proceed this possibility. One of several tasks, where NFC should be used as transportation unit, is the initial key exchange for pairing nodes with the gateway. These enhancements are important to secure the communication from the beginning of the first usage. All security enhancements should be combined together in an easy usable smartphone application for Android [68]. With the designed Android application the user can configure and read out the status information of the scanned IoT node. In addition, this application also performs the process of pairing new devices.

3.2 System Architecture

3.2.1 Topology

The network topology is a central point of defining the entire architecture of the smart home application. In a smart home application, it is important to distribute the nodes at these locations in a home where they are needed. Therefore, to be able to ensure that each IoT node has the opportunity to establish a connection with the gateway, different approaches have to be analyzed.

The first approach is to implement a hierarchical network structure. The gateway is located on top, which has a certain communication range. Inside this communication range several IoT nodes can be located to establish a direct communication channel. If a node is outside of this range, then it is not possible to establish a connection. For this use case it is required to extend the communication range with an additional node, which works as relay station. In this network topology, a relay station is responsible to forward data packages to the desired recipient. In Figure 3.2 illustrates the discussed network topology.

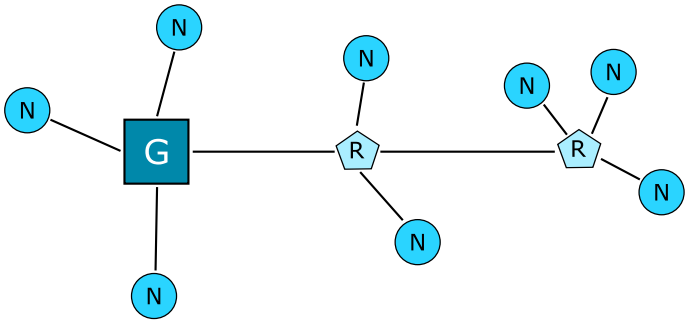


Figure 3.2: Hierarchical network structure with the top placed gateway (G) and its different communication participants. The communication participants are routers (R) and nodes (N).

The second approach is to construct a WSN with a mesh-network structure. A mesh-network enables the nodes to communicate with each other devices in a direct way. The advantage in this network structure is that each node can forward messages from other nodes to the desired destination. Consequently, the messages are routed to the desired

nodes over the shortest path. Another advantage of such a structure is that no further devices with special functions are needed and have to be placed in the right communication ranges to provide a full coverage. Some disadvantages exist for this network structure. One of these disadvantages is that each node would consume a little bit more energy because of the ability to receive all messages and for forwarding the message to the next devices. Figure 3.3 depicts the network structure of a mesh-network.

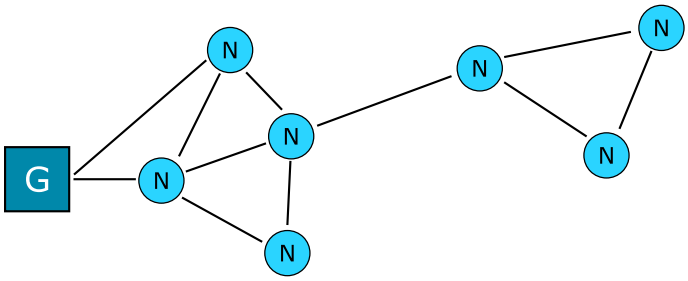


Figure 3.3: Mesh-network topology of arbitrary distributed nodes inside a WSN. The gateway is labeled with a "G" and the nodes with a "N".

3.2.2 WSN Communication Protocol

The choice of an appropriate communication protocol for the WSN was already predetermined due to compatibility reasons to other existing projects. The ability to communicate with existing environments should be possible in this master thesis. The overall project uses the 6LoWPAN protocol stack for the communication. Despite the fact of the fixed protocol stack, it has several advantages in comparison to other ones. An advantage of the 6LoWPAN stack is that the stack is a defined standard by the IETF working group and is free to use without the request of any license agreement.

6LoWPAN is placed on the top of the IEEE 802.15.4 standard, such as the other standard like ZigBee. The standard IEEE 802.15.4 defines the physical layer of the complete network stack. An advantage of the physical specifications is the usage of the ISM frequency band. From the ISM frequency band the typical frequencies that are used are 868/915MHz or 2.4GHz. These frequencies can be used without any admission or licenses, which make it easy to develop new devices in this frequency bands. Some regulations hold for these frequency bands, but they are only defining the maximum consumption of sending time in a specified time frame and the signal itself.

The usage of 6LoWPAN enables each IoT node to be directly addressed by its IPv6 address. The translation between the IPv6 or Internet Protocol Version 4 (IPv4) standard network to a 6LoWPAN wireless network is managed by a border router, which converts one physical platform into another one. The direct addressing of each IoT node enables several advantages like reducing the expense of supporting extra capabilities at each node. Another advantage for the usage of this network stack is the direct accessibility of node information and produced data, which can be requested directly from the nodes to use it for further proceedings. Instead of sending the request to a coordinator, which has to translate the request into the other addressing method, before sending it to the desired

destination. The reason for the described advantage is in the process of removing extra translation steps between different network protocols.

3.2.3 WSN Frequency Band

In a home automation application several frequency bands are used for the communication between wireless devices. Typically are used frequencies of the ISM radio band. These frequencies, which are not underlying a license, are: 433MHz (ISM-Band region 1), 868MHz (SRD-Band Europa), 915MHz (ISM-Band region 2) and 2.4GHz. Characteristic applications in a home automation, which are communication on these frequency bands, are for example wireless thermometers, alarm systems, wireless switchable outlets, and so on.

Mostly of the listed devices commonly uses the frequency band of 433MHz and 2.4GHz. Therefore, a reason for the decision to choose the frequency band between 863MHz and 870MHz was among other things the fact that this frequency band is not used so often. Another advantage in opposite to the higher frequency of 2.4GHz is the resulting efficiency in the usage of 868MHz for home automation because of the signal strength through the walls. The performance of indoor propagation is compared in the literature reviews [69] and [70].

3.3 Use Cases

The user has several entry points of using the designed smart home application. In this section the role of the user inside the IoT environment is analyzed. Various workflows are investigated, which are necessary for the application. For example, these workflows define where and how the user can receive information of connected IoT devices, how control commands can be sent to the devices, and the possibilities of interactions with the IoT nodes especially in a smart home application.

Figure 3.4 illustrates standard use cases, which a user has for a smart home application. This Figure additionally shows that the use cases can be grouped into three parts: devices which are connected via the WWW with the IoT environment, smartphone with NFC support, and the direct usage of implemented functions of the IoT devices.

3.3.1 Detailed Description of Use Cases

The uses cases, which are depicted in Figure 3.4, are explained in more detail in the next section. The use cases provide a good overview of the functionality of the overall system.

Devices with Active Connection to WWW

This group of use cases combines the functionality to display information of each IoT device on a website. An additional use case demonstrates that the user should be provided with the usability to send control commands to the devices. Control commands for IoT devices can be used to change the state of a relay, pushing a virtual button, or similar.

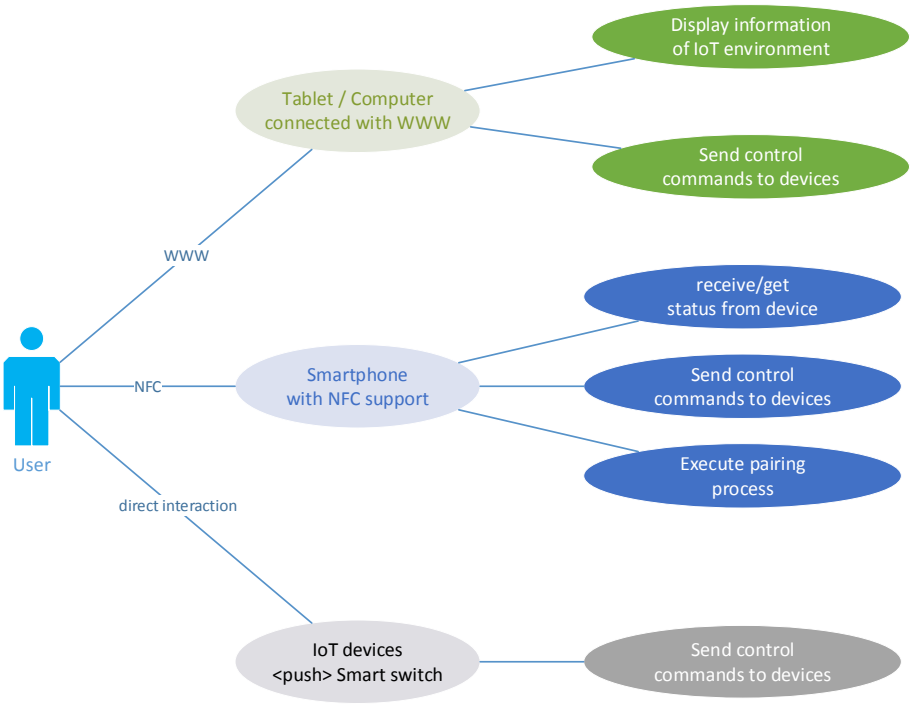


Figure 3.4: Use cases of receiving and sending information of IoT nodes in a smart home application.

Smartphone with NFC Support

The next group of uses cases relay on the support of NFC technology. The illustrated use cases have to be performed with an NFC-enabled device. The enhanced usage of this technology only receives the information of the scanned IoT device, which is in close distance. With the usability of NFC support, the functionality to send control commands should be provided to the devices, and to force the process of pairing nodes in connection with the key management system.

Direct IoT Device Interaction

The third group of the use cases discusses the cases which are available through the direct usage of functionalities of the IoT devices. Some IoT devices have external input channels for triggering actions. These input trigger signals can be created by the users through several actions, like a button press. One concrete example for this use case would be the push action of an external attached button of a smart switch. The depicted use case in the illustration shows exactly the described use case with the push action of a button. The result of the button press leads to sending a control command to the desired destination.

3.4 System Hardware Components

This section focuses on the designing and specification process of the complete IoT hardware environment. Each hardware component is analyzed on their requirements and the scope of usability. The IoT environment consists of two main hardware components: A central gateway and several IoT nodes with different functions.

3.4.1 Gateway

The gateway is the base station of the complete system and acts as router between the WSN and the WWW. The basic task of the gateway is to collect all information from the connected nodes and to prepare them for further processing. The received data is edited on the gateway and will be visualized on the self-hosted website. This website combines the received data with an interactive platform to be able to interact with the nodes. Some nodes have functions, which can be controlled by the user like switching a relay or pushing a button. Another essential task is the key management. The gateway is responsible to fulfill the requirements on the security aspects. In a typical smart home application, the gateway is located at a central and static location inside the home. Consequently, the restrictions on small construction size and low-power consumption are not as important as for the nodes in the WSN.

Components

The entire gateway device consists of different hardware elements to fulfill the specifications. Figure 3.5 depicts the main components of the gateway. The central element is the microcontroller, which is responsible to control all external devices and to execute the operations of the embedded OS. Furthermore, the construction includes a webserver, components to establish a connection to the WSN, a security controller, and finally a communications channel for supporting NFC technology.

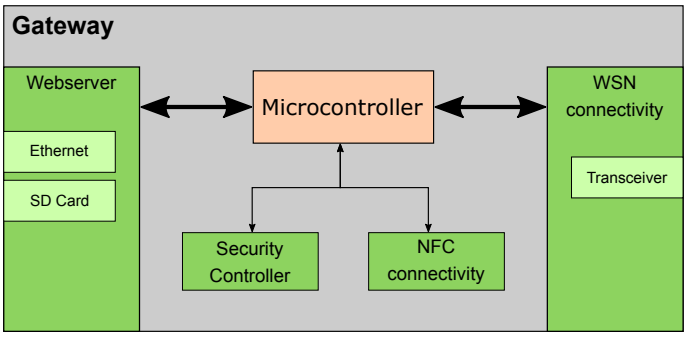


Figure 3.5: Hardware concept of the gateway with high level separation into main components.

Detailed description:

a) **Microcontroller**
The microcontroller is the central device of this hardware construction. Therefore, it is important to choose a device, which can fulfill all defined specifications. One possible device, which fits to the requirements, is the XMC4500 microcontroller from Infineon [71]. This microcontroller is equipped with a core element of an ARM Cortex-M4 Central Processing Unit (CPU) and provides many peripheral components. A good overview of all available components in the XMC4500 is illustrated in Figure 3.6. Essential components, which lead for choosing this microcontroller, are the integrated IEEE 1588 compliant Ethernet MAC module, support of Secure Digital Card (SD Card), the amount of several Universal Serial Interface Channel (USIC) modules, and large memory storage usability for the firmware.

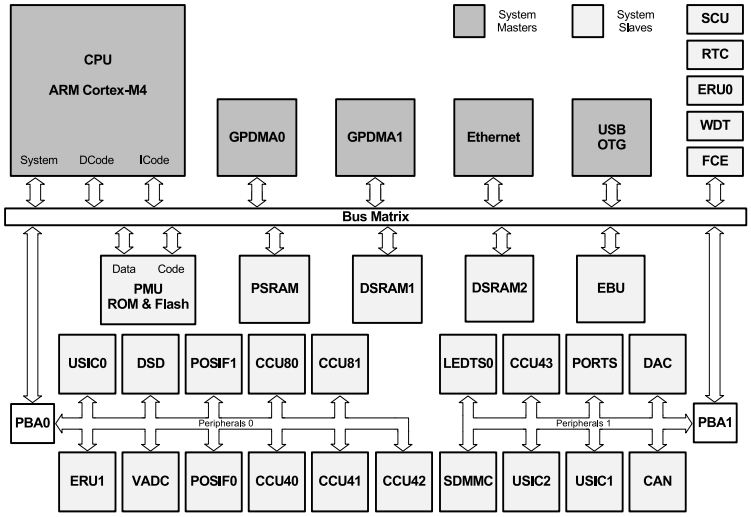


Figure 3.6: Internal system architecture of the XMC4500 from Infineon with all available components. Illustration taken from [71].

Key facts of XMC4500 [71]:

- 32-bit ARM Cortex-M4 CPU
- Maximum CPU frequency of 120 MHz
- 16 Kbyte on-chip boot ROM
- 1024 Kbyte on-chip Flash Memory
- 160 Kbytes Static Random Access Memory (SRAM)
- Floating Point unit
- Ethernet MAC module capable of 10/100 Mbits
- 6 USIC channels

b) **Security Controller**
Besides the microcontroller, the security controller is an essential part of the hardware composition. The reason for this importance is to provide a secured environment in the IoT network. In order to fulfill all needs, it is required to use a dedicated controller, which is especially designed for cryptographic tasks. In this thesis, the security controller is responsible on one hand to store data in a secured Non-Volatile Memory (NVM) and on the other hand to process cryptographic operations in a trusted environment. Stored data are a combination of configuration parameters of the gateway itself, cryptographic keys of paired IoT nodes, and specific device informational data. For the establishment of a trusted environment, some cryptographic functions like encryption, decryption, creating of signatures are required, including verifying signatures of transmitted data packets.

The selected security controller [1] provides a Universal Asynchronous Receiver Transmitter (UART) interface for the communication with the microcontroller. The interface itself uses the standardized ISO7816 standard. The security controller is constructed with a Java Card environment, which is expendable by developing of new Java applets. This characteristic of developing own applets for the security controller makes it an interesting product for the IoT environment because the configuration is consequently very flexible for different variations of nodes. The security controller also enables the usage of cryptographic methods inside a protected environment, allowing resistance against common side channel attacks.

c) **NFC**
The NFC connectivity is one of the essential extensions of the gateway. This extension allows to perform configuration operations even if the device is not connected to a power supply. In addition, this functionality provides the usability to exchange cryptographic keys in a secured and trusted environment. The technology for the NFC functionality is supplied by the selected security controller. Therefore, all configurations and data exchanges can be directly executed on the secured environment. The advantage of the integrated NFC support of the security controller makes it possible to transfer data even if the complete hardware device is not powered because the security controller can harvest the required energy out of the electromagnetic field from the readers NFC module.

d) **Ethernet**
The Ethernet module is required for hosting the webserver. The gateway should be connected with a network router to broadcast the website into the standard network infrastructure of a typical home. This module is essential for enabling the access to the distributed IoT network because it provides the connection between the WSN and the WWW. This Ethernet module is sufficient to support IPv4.

[1] Slightly modified internal Infineon test-chip, based on variants described at www.infineon.com/security

e) **SD Card**
On the SD Card storage should be placed the content files of the hosted websites for the webserver. The usage of an external storage for the data is recommended to save storage space in the more valuable ROM of the microcontroller. Another advantage of this design is the easy updating process of the content of the website.

f) **WSN Communication**
The gateway requires a communication channel to the distributed IoT nodes, which is provided through the usage of a dedicated transceiver module. Through the predefined specification of the frequency band, the used network stack, and the compatibility to other projects, it is suitable to use the SmartLEWIS transceiver with the exact label TDA5340 [72] for the communication. This transceiver enables the hardware platform to communicate with other desired IoT devices of the project. The TDA5340 transceiver supports several frequency bands, which is a useful property to operate in license free frequency bands. The interface to the microcontroller is established by a standard 4-wire Serial Peripheral Interface (SPI) module. This interface enables to control and access the buffers of the transceiver. Figure 3.7 illustrates the internal architecture of the TDA5340 device. This transceiver is very suitable for low-power devices because it supports different operating modes, which allows to reduce the power consumption to a minimum of $0.9\mu A$. Special functions of this transceiver module are the possibility to configure various channels with several operations modes, which can automatically listen in the frequency band for the desired message in a low-power mode. If the transceiver recognizes an assigned message, then the module wakes up and receives the complete message. This opportunity enables a highly sense for reducing the power consumption.

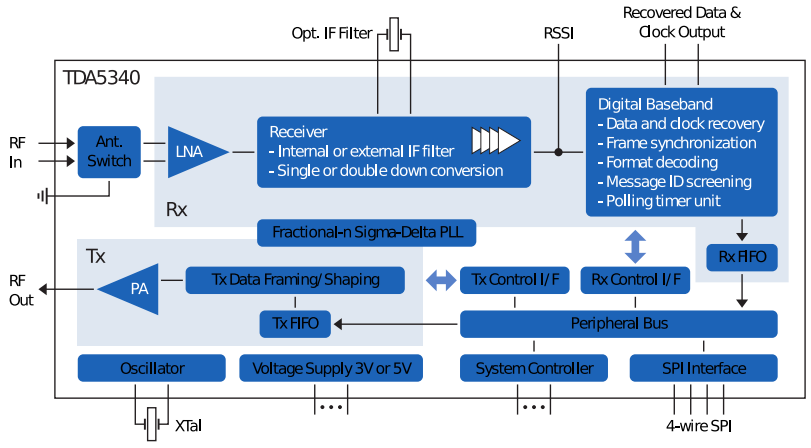


Figure 3.7: Internal architecture of the SmartLEWIS - TDA5340. Illustration taken from [73].

3.4.2 IoT Nodes

In a WSN exists several nodes, which can be carried out as a sensor or as an actuator platform. The node is called IoT node in the thesis, due to the reason that it is specialized

for operating in a bigger environment including the integration of IPv6. In a smart home application are distributed the nodes at the places where they are required to make a home smarter. Due to this reason, the exact places for installing the IoT nodes depend on the implemented functionality. Some useful products for a smart home application are: smart outlet, smart switch, weather station, alarm system and so on. The IoT/WSN node is a redesign based on an early prototype version developed at Infineon.

In the design process of this IoT node, the construction is separated into two parts: main board and extension board. The main board should be equipped with all necessary components, which are required to operate inside a typical WSN. The basic functions of a node are to receive and transmit data inside the wireless network and to provide hardware protected security features. The extension board specializes the IoT node with hardware components of sensors and actuators. In the end, these additional hardware components define the provided functionality of the IoT nodes. Furthermore, in the design process should be considered the limitations of power consumption and construction size.

Basic System Components

Figure 3.8 illustrates the main components of a standard IoT node. With these main components it is possible to communicate inside the WSN and to operate in the entire IoT environment. In addition, the integration of a security controller enables the opportunity to secure the communication between devices.

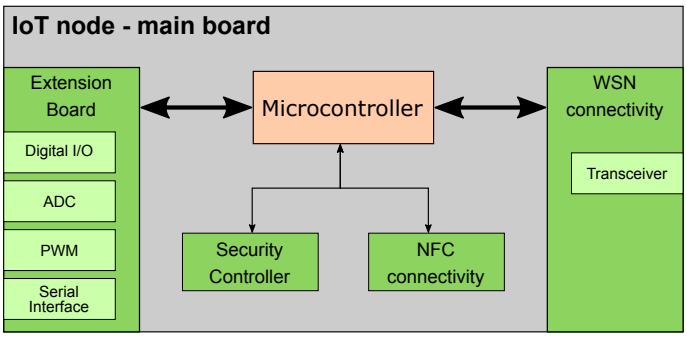


Figure 3.8: System architecture of the main board of the IoT node with all essential components for a basic operation.

Detailed component description:

- **Microcontroller**
  On this hardware platform the central hardware component is as well the microcontroller. This microcontroller should provide all opportunities of interfaces for the requested hardware components, in order to process all tasks. In the end the interfaces of the microcontroller should be exploited in a high level. This fact is important to provide all functionalities on a tiny footprint. Low power consumption is also considered in the selection of a microcontroller. In general, a low-power consumption is important among other things for an IoT device in order to increase

the lifetime with only one battery charge. The common tasks for the microcontroller are to control the wireless communication channel, the security controller, and all external hardware components of the extension board. One suitable microcontroller for these requirements is the XMC1100 [74]. Some key facts of the internal modules of this microcontroller are: two USIC channels, several Pulse-Width Modulation (PWM) generation outputs, Analog to Digital Converter (ADC), and an Event Request Unit (ERU). All internal available modules of the XMC1100 are depicted in Figure 3.9.

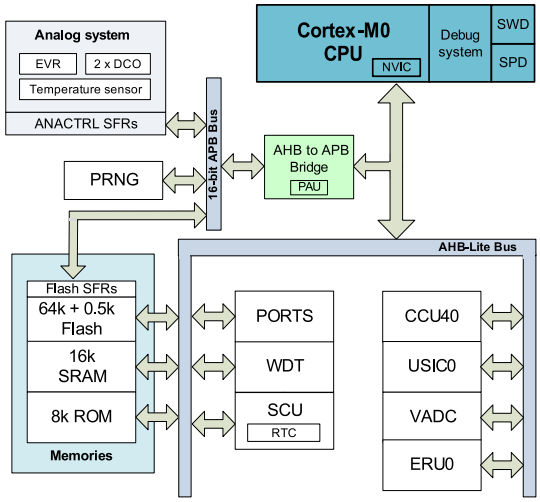


Figure 3.9: Internal architecture of the XMC1100 with its available modules. Illustration taken from [74].

- **Security Controller**
  The used security controller is the same as for the gateway (see Section 3.4.1). The basic tasks of this security controller are to store used cryptographic keys, node specific data, and configuration parameters in a secured NVM. Another necessary task is to provide hardware supported cryptographic functionalities to the microcontroller. The used security controller is additionally equipped with NFC technology. NFC is used by the platform as communication channel to force the process of pairing and to exchange configuration data.

- **WSN Communication**
  For wireless communication with the gateway, it is required to have a radio unit to provide access to the WSN infrastructure. Consequently, the same transceiver module is used as well as for the gateway, namely the TDA5340. This module requires an external antenna to operate in a normal condition. In order to reduce the construction size of the entire node, the focus is lead to use a printed antenna design instead of attaching an external one, which requires a lot of space.

- **Connectivity to the Extension Board**
  The main board has to provide a variation of different interfaces for the extension board. Some useful interfaces for offering a wide spread opportunity to connect

various sensors and actuators, are for example SPI, Inter-Integrated Circuit (I^2C), digital input/output pins, ADC input pins, and PWM outputs. The wide diversity of different interfaces ensures that the IoT nodes can be extended with a lot of different functionalities.

Extension Boards for the Main Board of the IoT Nodes

The extension board, which is connected to the main board, defines the type and provided functionalities of the IoT node. The established interface to the extension board contains a variety of different connection opportunities, which are discussed previously in the basic board definition. The first step in turning a typical home into a smart home is to extend a normal outlet with an IoT node. The result of this extension is a smart outlet device, which provides several functionalities. The same procedure is applied to a typical switch for lights. With these two improvements of standard components, a general home turns into a smart one, due to controlling the outlets by the IoT environment, and to collect additional data for cross-platform controlling processes.

In the following section is focused on the design process of two extension board platforms for a typical smart home application:

- **Extension Board for a Smart Outlet Application**
  The device of a smart outlet extends a normal wall outlet with additional functionality by adding an IoT node. This new developed node should be able to turn on and off the power supply of the connected device. Furthermore, the node adds special functionalities to the outlet like measuring the latest current consumption and the attached voltage. With this information, the IoT node can provide useful information about the connected device to the user. This information is useful to understand the power behavior of connected devices at the smart outlet. In the design process, it should be ensured that the resulting construction size is as tiny as possible, in order to fit behind a typical wall outlet. The result of this extension board is to extend each existing outlet in a home with smartness. Figure 3.10 depicts the hardware elements, which are required for the realization of this IoT node.

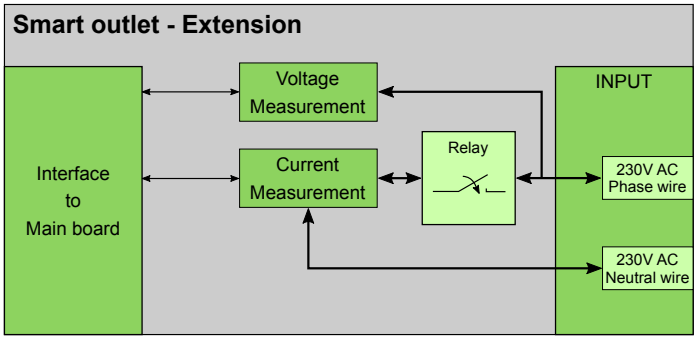


Figure 3.10: Architecture overview of the extension board for a smart outlet application.

Used components:

- **Current Measurement**
  The current measurement unit has the task to sense the latest current, which flows through the attached outlet. In a typical home with an electric installation, it is possible to flow a maximum of 16A through the outlet. The selected sensor for this task is the TLI4970 [75], allowing to measure the current in a galvanic isolated method. The sensor, TLI4970, supports a current measurement range of +/- 25A. The accuracy of the measurement is correlated with the precision of the integrated ADC, which has a resolution of 13 bits. For the exchange of the measured data, this sensor uses a standard SPI interface. In addition, this module also supports a fast over current detection, and provides an opportunity to update the internal configured filter settings for the measurement process.
- **Voltage Measurement**
  The process of measuring the voltage of the attached power supply is an essential point for the IoT node, in order to calculate reliable power consumption of the connected device. Consequently, the supply voltage of the outlet should be measured with an appropriate designed circuit. In the implementation process, it should be analyzed if it is necessary to implement an isolated measurement, or it is sufficient to measure it directly. In principle, this IoT node, which is executed as smart outlet, has no physical connection to the outside materials. Consequently, the circuit for the voltage measurement has not absolutely be constructed in an isolated process.
- **Relay**
  The power supply of the smart outlet should be able to be switched on and off in order to control the attached device. For this task, it is required to use a relay. Different variants are available, which directly affects the construction size of the platform and the maximum switchable power consumption. The maximum possible current which can flow through a typical outlet are 16A. This high current value is not easy switchable in a small construction. Therefore, since the node is currently only intended for basic demonstrator purposes, a small form factor of the relay was desired. Consequently, for this thesis a solid state relay is selected, which is the AQH3213A, which supports a maximum current flow of 1.2A at 600V Alternating Current (AC). The advantage of using this solid state relay is the small construction size and on the automatic function of switching the contacts when in the AC supply voltage a zero crossing is detected.

- **Extension Board for a Smart Switch Application**
  This extension board enhances a normal light switch into a smart one. The smartness of this switch leads to a free configurable pairing of switchable end devices. The main function of this smart switch IoT node is to identify the press of the attached switch in order to change the output of a paired IoT device. The analyzed input value is transmitted as execution command to the gateway station or directly to the paired device. A simple overview of the designed architecture of the smart switch extension

board is illustrated in Figure 3.11.

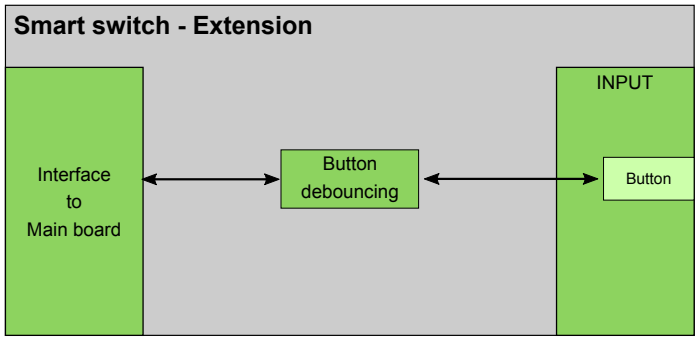


Figure 3.11: Architecture overview of the extension board for a smart switch application.

Used components:

– **Button Input**
The button is the only external input interface to this extension board and should be tracked by the microcontroller. The input signal of the button should be debounced in hardware or software. In addition, the microcontroller is supposed to use as input an input pin with the support of an interrupt enabled module. With the support of this interrupt module, the microcontroller does not have to poll the status of the connected switch all the time, in order to detect changes. Consequently, the microcontroller can be in a sleeping modus until the switch is pressed and as side effect is reduced the energy consumption.

3.5 System Security Architecture

In a smart home application, it is important to implement an adequate security architecture in order to ensure the privacy of every household. The privacy of a single household can only be achieved by combining cryptographic methods, which are implemented at different parts of the entire structure of the IoT environment. Another focus of this work is to focus on enhancing the security of the IoT environment, especially the communication inside the WSN. The implemented security should be envisioned up to the gateway. In the part of the WWW it would be necessary to implement a webserver with extended security features like Transport Layer Security (TLS) or similar, but these features are out of scope of this master thesis.

3.5.1 Composition

The overall network channel of the WSN should be encrypted with a separate network key. This procedure is conducted to hamper the attacker from eavesdropping the entire network traffic. The central communication station, which is the gateway, acts as a trusted center in the IoT network. This means that for all IoT nodes, a secured communication channel to the gateway must be established, resulting in each IoT node having a secured point-to-point communication channel to the gateway. This point-to-point communication

is essential for transferring critical or confidential information between two participants. Confidential information could be for example updated cryptographic keys, node status information and sensor values which are used as input for additional electronic devices. All cryptographic mechanisms, which can be implemented in this architecture, have to be analyzed regarding the generation of overhead for the communication and the execution time on the microcontroller. One essential requirement of the complete IoT environment is to still be energy efficient. Therefore, a trade off has to be considered between the key length and the resulting encrypted payload length. A long resulting payload leads to a longer transmission time between the IoT nodes. A longer transmission time also conducts a higher energy consumption, which should be kept as low as possible.

Network Encryption

The network layer encryption should be light weight and fast for encryption. A common way for this encryption operation is to use AES with a key length of 128 bits. All communication participants inside the IoT network share the same AES key, due to the symmetric cryptographic algorithm. The network encryption is the first step of securing the network infrastructure. The result of implementing a link layer encryption would make it possible to transmit insensitive information in plaintext inside the protected network. Nonetheless, for very sensitive data it is still important to only send encrypted data packets because it can happen anytime that a node is attacked and taken over by an attacker. If the attacker has full control over the node, it would be possible to eavesdrop unprotected communication inside the network infrastructure, resulting in a security breach.

Point-To-Point Encryption

The point-to-point encryption between the communication participants is required to enable privacy among them. If the network encryption is broken due to a malicious node in the entire network, then the transmitted data is still protected by the point-to-point encryption. Furthermore, this encryption process authenticates transmitted payloads. This authentication verifies if the transmitted payload has been modified during transmission. Therefore, it is required to have two independent cryptographic keys for one point-to-point encryption: One for the payload encryption, and one for the authentication process. When selecting a suitable cryptographic algorithm holds the same limitations of energy saving mechanism than for the network encryption.

These requirements lead to the consideration of two different cryptographic techniques: Symmetric and asymmetric cryptography. Symmetric cryptography shares the same key between the paired devices. A common symmetric cryptographic algorithm is AES [76] in combination with a block cipher. The other possibility is the usage of asymmetric algorithms. For example, public cryptography belongs to the group of asymmetric algorithms. For this group typical algorithms are Rivest-Shamir-Adleman (RSA) [77] and ECC [77]. The advantage of ECC is the shorter key length in comparison to RSA. In Table 3.1, the different key lengths in bits are compared, which are required to reach the same security level. The conclusion is that currently, ECC can satisfy with a shorter key length a higher security level, than in comparison to RSA. A shorter key length also means to require less storage space for the cryptographic keys in the secured NVM.

Security bits	Symmetric encryption algorithm	Minimum size (bits) of public key	
		RSA	ECC
112	3DES	2048	224
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

Table 3.1: Security comparison of various algorithm-key size combinations. Information based on [78].

For point-to-point encryption, a security level higher or equal of 128 bits should be used. Row two of Table 3.1 should be used as guideline for this master thesis. The next section is focused on the two different combinations of cryptographic algorithms, which can be used in the WSN for encrypting the transmitted messages. The cryptographic approaches with AES and ECC will be discussed in the next section. Excluded in this analysis is RSA, due to the long processing time of creating new RSA keys with hardware support in the embedded world. Another disadvantage of RSA is the long resulting key length in order to reach a similar security level. In the design process, it should be considered that for each transmitted message the integrity of the message should also be ensured. For this operation, MAC algorithms are usually used.

- **Authenticated Encryption [79] with AES**
  Authenticated encryption with the usage of AES is separated into two parts. The first part is to encrypt the plaintext, and the second part is to calculate the MAC. Both use a CBC mode in combination with an AES cipher. Figure 3.12 depicts the named CBC mode with the used AES encryption as cipher function. The AES cipher is used with a block size of 128 bits. Therefore, the following parameters are required for each encryption process: the cryptographic key for the AES modules, a randomly chosen Initialization Vector (IV), and the plaintext. The plaintext is separated into blocks of 128 bits before it can be used for the AES ciphers. After the finished encryption process, the encrypted message blocks of 128 bits can be transmitted to other paired participant. In the transmission is included the encrypted message, includes the randomly chosen IV in plaintext.

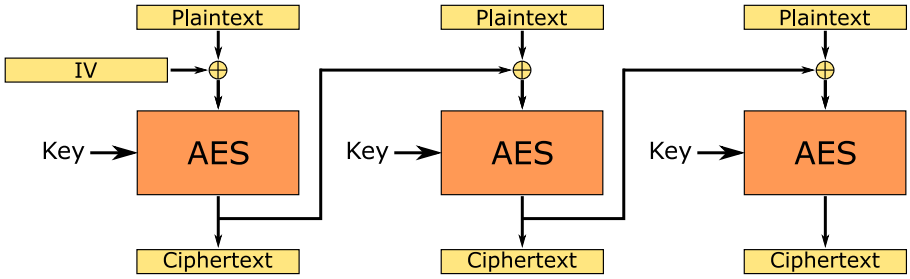


Figure 3.12: Encryption process of a CBC mode with usage of AES cipher.

In the process of generating the MAC, the CBC mode can be used as well, which is illustrated in Figure 3.13. The only difference in the usage is the fixed value of the

IV. The IV can be set to zero, during the calculation of the MAC. Consequently, the following parameters are required for this processing: a 128 bit AES key and the input message blocks of 128 bits. The resulting MAC is the output value of the last processed AES cipher block, which means that the value of the calculated MAC is 128 bits long.

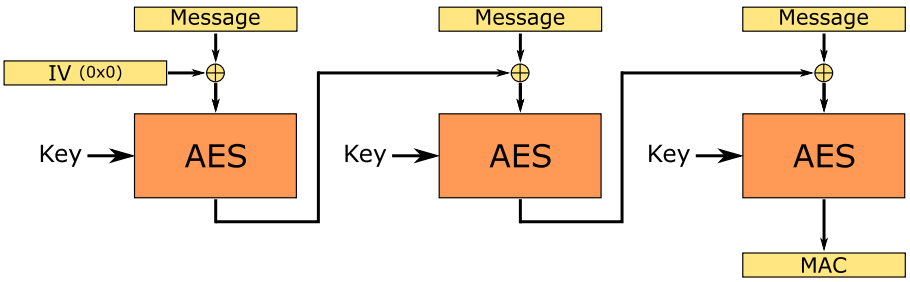


Figure 3.13: MAC generation in CBC mode with the usage of AES cipher.

The processed output value of encrypting and authenticating a plaintext, consisting of multiple blocks with a block size of 128 bits, results into a minimum total size of 384 bits.

Detailed breakdown of data length:

- 128 bits IV
- 128 bits * n (n = number of encrypted block messages)
- 128 bits (MAC)

- **Authenticated Encryption with ECC**
  It is not possible to encrypt a plaintext directly with the support of ECC. Therefore, special methods like Elliptic Curve Integrated Encryption Scheme (ECIES) for the encryption process have to be used [80]. Figure 3.14 illustrates the functional diagram of the ECIES concept for the encryption process of a plaintext. This procedure is split into several activities and is described in detail in the next section.

  The several processing steps in an ECIES encryption process are labeled in Figure 3.14:

  1. In the first step of the process, the initiator creates new temporary keys out of the private/public key pair of ECC. The new ephemeral private key is denoted as u, and the temporary public key as U.
  2. In step two, the initiator uses the Key Agreement (KA) function, in order to process a shared secret. The shared secret is the product of the initiator's temporary private key and the recipient's public key V.
  3. Now, the initiator takes the shared secret as input data with optional parameters for the Key Derivation Function (KDF). The output value of this function is then the concatenation of the new encryption key and MAC key.

4. In step four, the initiator can use the first part of the processed KDF output keys, k_{ENC}, as input for the symmetric encryption algorithm. The encrypted message is denoted in the Figure as c.
5. Next, the encrypted message is taken as input for calculating the MAC with the second part of the processed key, k_{MAC}.
6. After these steps, the initiator provides the encrypted message to the recipient with the concatenated of these values: $U||c||tag$.

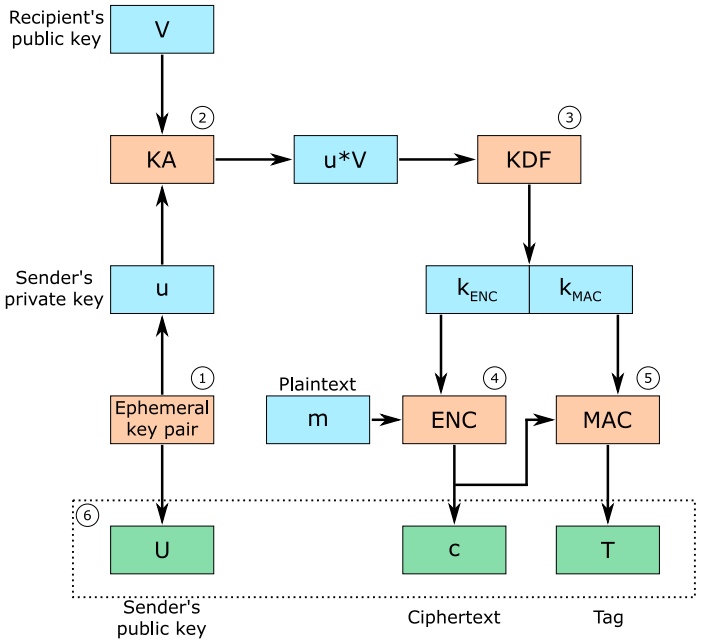


Figure 3.14: Functional diagram of an authenticated encryption process with ECIES. Illustration based on [81].

Due to the required key length of 256 bits for ECC, information taken from Table 3.1, the public key U results in a total length of 512 bits because the public key of ECC is represented as a point on the elliptic curve and consists of two parameters x and y. Each parameter has a length of 256 bits. The resulting output of the encrypted message is a composition of multiple blocks with a block size of 128 bits. The tag size of the used MAC is also 128 bits long. Consequently, the minimum length of a successful encrypted and authenticated message is 768 bits.

Detailed breakdown of data length:

- 512 bits ECC public key
- 128 bits * n (n = number of encrypted block messages)
- 128 bits (MAC)

The conclusion of the comparison between AES and ECC is that the entire packet of one encrypted message is shorter with the usage of AES encryption instead of ECC. The

reason for this is the created overhead of the ECC with a public key length of 512 bits. As previously mentioned, a public key for ECC consists of two parameters with a length of 256 bits for each parameter. Therefore, in the point-to-point encryption process an AES cipher in combination in a CBC mode is used.

3.5.2 Key Management

Another essential part in the security architecture of an IoT environment is the key management system. The support of transmitting encrypted messages inside the IoT network is not enough to result in a secured environment because static keys could be figured out sometimes and would lead to a broken encryption scheme. Hence, the responsibilities of the key management are to provide features like node revocation, collusion resistance, and resilience. Key management can additionally be combined with an intrusion detection system, in order to detect malicious IoT nodes in the existing environment. This combination allows the key management to exclude attacked IoT nodes without disturbing the complete network infrastructure. Another important point is to control the key freshness of the distributed keys, in order to make it more difficult for the attacker to find out the present distributed keys. Sensitive or critical information of the key management system, such as all cryptographic keys, have to be stored in a secured NVM. In addition, the secured NVM is protected against side channel attacks, which protects the keys from being extracted out of the hardware device, even if the hardware device is in the hand of the attacker.

In an IoT environment, it is essential to be supported by a dynamic key management system, which can deal with adding and removing of different IoT nodes. One key part of the key management system is also the process of pairing new devices with an accurate security level. The complex process of pairing new nodes to the infrastructure can be split into several tasks. Some parts of these tasks can be automated by the security controllers, and some have to be processed outside with the interaction of the users.

Concepts

The selection of an appropriate key management system depends on various characteristics. These characteristics relate to the following topics: the chosen network structure, distributed security architecture, operational costs and others. The key management system should be lightweight and energy-efficient. As previously discussed, the devices are connected within a star-network topology in the WSN. A star-network architecture for a key management system is easier to handle than for a mesh-network because each node is directly connected to the gateway with an encrypted channel and the rekeying process can proceed in a direct way. A new IoT node must be paired for the first usage in the network. After a successful pairing, the node can communicate with the trusted center over a secured channel. If the nodes want to exchange data with another node, then the trusted center should provide methods to create a separate secured channel between these two participants.

For the used network structure, it is suitable to use a centralized key management system because the gateway should manage the complete network communication in the IoT

environment and the gateway has no limitation in power consumption. Another advantage in the design of this thesis is that all participants in the IoT environment have a dedicated security controller. With the usage of the security controller on each node, new cryptographic keys can be created directly on the controller with the support of hardware accelerations. This means that each hardware platform has the possibility to create on their own new keys and to directly store the keys in a secured environment, without a bypass over unprotected paths.

A list of various suitable concepts of key management system for the described requirements are:

- Flat network based
- Hierarchical network based
- Heterogeneous network based

One of the targets of this master thesis is to demonstrate the advantage of the support of NFC-enabled devices. With the support of NFC-enabled devices, it is possible to configure and pair new IoT devices for an IoT environment. Due to this reason, a flat network based key management system is convenient for this application. The reason for this is that each IoT has to be previously configured with cryptographic keys to be able to communicate inside the network. One key management system is KeyRev [43], which meets the basic requirements. The other key management concepts, hierarchical and heterogeneous based ones, are for the first implementation too complex for the desired management system. One reason for it is the way of distributing the keys to new nodes, which in the hierarchical based system is managed over the wireless communication channel. In a heterogeneous based key management system, different types of nodes are used to distribute the individual keys, which means that the IoT network would have to be extended with several devices. These new devices are required for managing the key distribution system in a heterogeneous based system. In conclusion, the designed key management system is based on the KeyRev scheme (explained in Section 2.4.2) with slight modifications to fulfill the needs of this thesis.

Major Functionalities

This section is focused on the requested needs to the key management system and to point out the main functionalities. The key management is responsible to handle different processes like: generating new cryptographic keys, storing of created cryptographic keys, defining the process of pairing nodes, and renewing of paired information in relation to security. On each IoT device, the basic functions of the key management system are implemented with several extensions. The amount of distributed functionalities depends on the type of the end device because the gateway as a trusted center requires more key management functionalities than a simple client, like the distributed nodes.

- **Management of Cryptographic Keys**
  The key management system on each IoT device should be able to manage the process of creating new cryptographic keys for the encryption processes. It is also responsible for storing the created keys in a secured environment in order to hamper the extraction of keys.

- **Initialization Process when Integrating a New Device**
  The workflow of the initialization process of integrating a new IoT device has to be established with the support of NFC technology. This procedure enables a new perspective of the key management system, which focuses on generating a secured environment from the beginning of the first usage. The detailed workflow of pairing IoT devices with each other have to be defined and fixed in the process of the key management system. In general, two different workflows are required to be defined, in order to handle the pairings between the participant combinations of a gateway to a node and between two nodes.

- **Process of Updating Keys for the Network Layer Encryption**
  The network layer encryption key can be distributed to all participants in a secured environment by sending the new key through the existing point-to-point encryption channel, which has been established between each IoT device and the gateway. If the gateway recognizes a malicious node by an intruder system, then it can be easily excluded from the protected environment because the malicious node cannot eavesdrop the new key of the network layer encryption. The event for renewing this key has to be forced by the gateway. The new network layer key is created with the support of the security controller on the gateway side and is afterwards distributed through the secured channels.

- **Process of Updating Keys for a Point-To-Point Encryption**
  The update process of this point-to-point encryption channel is constructed in an easy way because the old encrypted channel is used to transfer the new created pair of cryptographic keys. The updating process for this use case should be forced from the node side to the gateway. This means that each IoT node is responsible for creating their new cryptographic keys in a secured environment and will finally be transmitted to the gateway to renew the updated key management information.

Distributed Keys Inside the WSN Environment

By the use of the desired cryptographic structure, each device is responsible to manage a different amount of cryptographic keys. Table 3.2 describes the number of used keys for each IoT device. As Table 3.2 demonstrates, the distributed IoT nodes have to store a minimum of three different keys for a secured communication between the gateway. Additionally the number of keys can be increased by the amount of two, when the node is able to be paired to another IoT node. The gateway has to be designed in a way that makes it possible to store many keys. The reason for this storage option is because the gateway has to be able to communicate with each node in the IoT environment and consequently requires two separated cryptographic keys for each communication channel.

Device	Number of keys	Cipher algorithm	Direction	Used keys
Node	1	AES-128	Network wide	Network layer key
	1		Gateway - Node	Encryption
	1			HMAC
	1 (optional)		Node - Node	Encryption
	1 (optional)			HMAC
Gateway	1	AES-128	Network wide	Network layer key
	# of paired nodes		Gateway - Node	Encryption
	# of paired nodes			HMAC

Table 3.2: Detailed overview of distributed keys on various IoT devices. The IoT environment only consists of two devices, namely a gateway and nodes.

Each device is responsible to manage and store the different cryptographic keys in a secured way. This means that the keys should be stored in a special NVM, which is resistant against side channel attacks and other security analysis.

Pairing Procedure for Integrating New IoT Devices

The pairing process in the key management system is an important point because this is the point where the communication channel changes from an insecure state to a secured one. In general, the process of pairing means to exchange cryptographic keys between two participants. It should not be possible for an attacker to interfere in the process, or steal the keys. Therefore, it is not recommended to transfer the keys over an unprotected channel to the other participant.

In order to overcome the described problem, the procedure should be introduced where the cryptographic keys can be exchanged over a protected channel. One possibility is to use the NFC integrated option of the security controller to exchange the keys between the different participants. A mobile NFC-enabled device, like a smartphone, can be used to proceed the pairing with an easy workflow for the users to close the unprotected gap between the two participants. The advantage of this procedure is that each device in the IoT environment can be configured and pre-loaded with the cryptographic keys, before they are used for the first time. This scenario is possible because NFC supports the powering of another device by its electromagnetic field. With this opportunity it is possible to transfer the keys in a secured and trusted environment.

In this thesis, two different workflows for pairing a new device to the IoT network are defined. The workflow that is selected for the pairing process depends on the types of the two pairing participants. For the pairing process between an IoT node and the gateways exists a workflow and also an additional one for the process of pairing two IoT nodes. These workflows only differ in the amount of exchanged cryptographic keys, and also in their exchange process. The detailed information on the amount of keys which have to be exchanged, was previously defined in section 3.5.2. Consequently, the communication channel between the gateway and a node requires to transfer three different cryptographic keys, in order to change the channel to a secured one. In order to complete the process of

securing the communication channel between two nodes, only two different cryptographic keys are required.

- **Procedure of Pairing Between a Node and the Gateway**
  Figure 3.15 illustrates the basic steps for the first pairing of a node with a gateway in the IoT environment. Following the steps, the user has to proceed with an NFC-enabled device. At first the user has to request the link layer encryption key of the gateway, therefore the user has to scan the gateway with the NFC-enabled device. On the mobile device, the cryptographic key is temporarily stored. The next step for the user is to scan the desired IoT node to transmit the link layer key and to receive new cryptographic keys for the point-to-point encryption. The scanned IoT node creates new cryptographic keys for each new pairing process inside of the dedicated security controller using True Random Number Generator (TRNG). The third and last step is used to transmit the previously received cryptographic keys from the IoT node to the gateway. After these steps, the gateway is securely paired with the IoT node and is ready for the common operations. The separate communication channel between the smartphone and the IoT devices also has to be secured with an adequate cryptographic algorithm. One suitable solution for this operation is the usage of public cryptographic algorithms, like ECC.

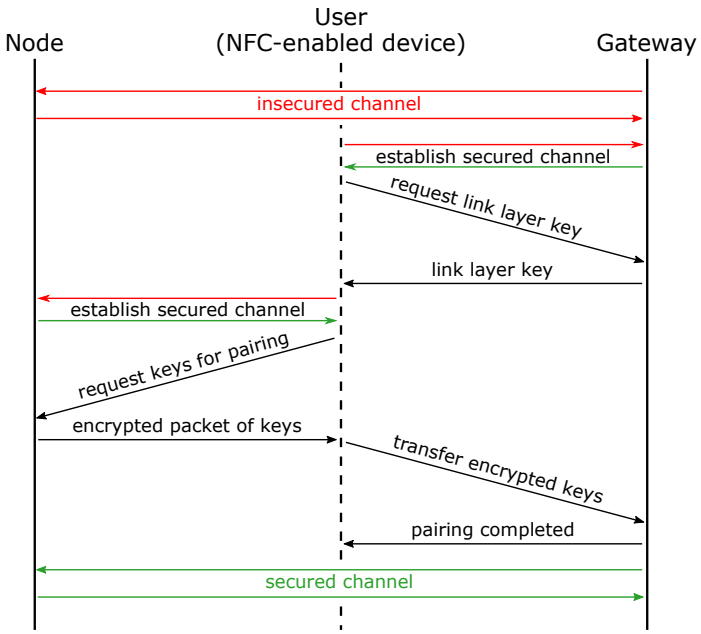


Figure 3.15: State diagram of required steps for proceeding a secured pairing process from a node to the gateway.

- **Procedure of Pairing Between Two Nodes**
  The process of pairing two nodes with each other requires a separate workflow because the amount of exchanged cryptographic keys differs from the process of pairing with a gateway. This pairing is required to obtain a point-to-point encryption between two desired IoT nodes. The extra encryption between two nodes is constructed

to transmit very sensitive or critical data over several nodes. The advantages of a direct encryption model are to save the amount of encryptions and the time of transmission. These savings lead to a general reduction of power consumption.

This additional pairing can be implemented in different ways. One way would be to use a similar pairing process than for the gateway with some revisions. The detailed workflow of the pairing process is illustrated in Figure 3.16. The first step in this process is to scan the first node. During this action the new node creates new cryptographic keys for the new point-to-point encryption channel and responds to the smartphone with the new keys as payload. In the second and last step, the desired IoT node has to be scanned to transfer the temporary stored cryptographic keys to the node. After these two steps, the two IoT nodes are successfully paired with each other and the nodes can exchange information through the secured point-to-point connection.

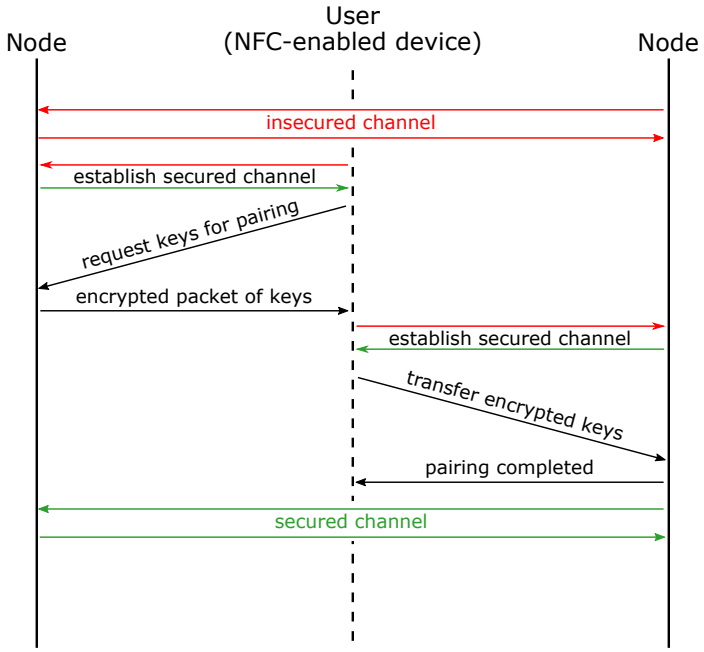


Figure 3.16: State diagram of required steps in order to proceed a secured pairing process between two nodes.

3.6 System Software Components

3.6.1 Smartphone

The smartphone plays a central role for the pairing process of the devices in the IoT environment. Nowadays, smartphones are increasingly equipped with the support of NFC technology. Therefore, it is possible to simplify the procedure of pairing for the user with the help of the NFC technology. The reason for the simplification is that the user only has to make one movement, namely the action of scanning the selected device with their smartphone. The own designed application on the smartphone coordinates the required tasks for performing a correct pairing. These operations include the exchange of the cryptographic keys for the encryption process or to update configuration settings. In order to simplify the workflow for the user, an Android application should be developed which follows the key exchange processes and guides the user with easy understanding instructions.

From the perspective of the user, it is important to use the least number of steps for performing a pairing process. The process of pairing should be as simple as possible because when the activation of security enhancements costs too much time or is too complicated, a wide range of users will not use it. A good implemented security architecture is not useful when it is not activated in the normal operation. In addition, the smartphone should display information of the IoT nodes, which are scanned with the support of NFC. With this feature the user is able to request current status information of diverse nodes in a directly way.

Requirements

For the pairing process and configuration routines, an application for smartphones with an Android OS should be developed. The application should be easy to handle and offers the user an assistance for the different task sequences. One of the important operations is the execution of the pairing process of two devices in subscription to exchange cryptographic keys. The pairing process, as previously defined, is split into two workflows: one for the combination of a node with a gateway, and the second one is constructed for two nodes.

Key Requirements:

- Show status information of nodes and the gateway
- Pairing procedure between a node and the gateway
- Pairing procedure between two nodes
- Send control commands to nodes over NFC

Android Application

The application is structured into four different activities: Main activity, Pairing between node and gateway, Pairing between node and node, and sending control commands to the IoT devices. In Figure 3.17, the different activities with their rough calling flow are depicted.

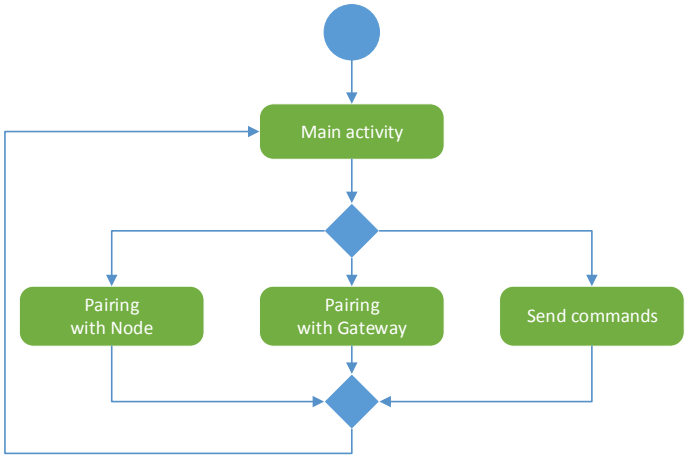


Figure 3.17: Concept of Android application with the combination of different activities.

The depicted activities of Figure 3.17 are described below in more detail with their included functionality:

- **Main Activity**
  The main activity has the responsibility to receive and display the information of the IoT devices. The information of the nodes contains basic data (address, type, and security enabled flag) and sensor/actuator specific data. This information should be received with the support of NFC technology by scanning a desired IoT device. After the received type information of the scanned device, additional buttons are to be displayed in the Graphical User Interface (GUI), with actions for sending specialized control commands, and for starting the various pairing processes.

- **Pairing Between Node and Gateway**
  In the pairing process between a node and the gateway, the user has to follow different steps for a successful completion. These steps have the function to transmit different cryptographic keys between the node and the gateway in a secured environment. The amount of transmitted cryptographic keys and the detailed specification of the keys are previously defined in the subsection "Pairing procedure for integrating new IoT devices" in Section 3.5.2. The additional communication channel between the devices and the smartphone, which is established with the support of NFC, should be protected by cryptographic functions to provide a trusted environment.

- **Pairing Between Node and Node**
  The process of pairing between two nodes is very similar to the gateway and only differs in the amount of transmitted keys. Furthermore, one node can only be paired with another node and do not support multiple pairings at the moment. Due to this limitation, each execution of this pairing procedure overwrites old pairing information.

- **Send Control Commands to Node**
  Each IoT node is defined with special functionalities. Consequently, each node type has different implemented functions, which can be performed and triggered from outside. With this Android activity, the user should be possible to execute these functionalities on the node by scanning it with the smartphone. This feature allows the user to directly control the IoT device by scanning it with the NFC-enabled device.

Security

The NFC communication channel has to be protected with cryptographic methods in order that an attacker cannot easily interfere the communication or eavesdrop the transmitted sensitive information between the IoT device and the smartphone. A solution for this problem is the usage of public key infrastructure with ECC. This infrastructure creates session keys, which are only valid for a certain time. These session keys are created with the ECIES method. The ECIES method has been explained in detail in Section 3.5.1.

3.6.2 Embedded OS and Enhancements

Contiki OS is an entire embedded OS, with the possibility to use it out of the box for standard use cases. In the use case for this master thesis, it is necessary to adopt and extend the structure of Contiki, in order that Contiki can operate with their own developed hardware platforms and software architectures. The base structure of Contiki OS is depicted in Figure 3.18. Contiki OS is clearly structured into several folders. Each folder groups files with the same functionality. The depicted structure has to be extended to the needs at several components like CPU, platforms, and additional hardware devices. The next section discusses the parts of the Contiki OS that have to be modified and extended.

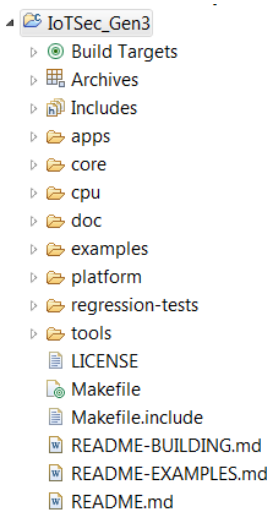


Figure 3.18: Basic file structure of Contiki OS.

Required Enhancements on Base Structure

The base structure of the Contiki OS only includes a small selection of implemented platforms and CPUs. In this master thesis, official maintained microcontrollers are not used and therefore they have to be added to the existing structure. New microcontrollers are placed in the implementation process in a subfolder inside the "cpu" folder.

Due to the fact that new hardware platforms are designed for this thesis, they also have to be added to the existing structure. The hardware platforms are added into the existing folder "platform" of Contiki OS. The platform configuration contains all important information about the each used hardware component, pin configuration of the microcontroller, interconnectivity between the devices, and the general system configuration.

On the new designed hardware platforms, new hardware components like the current sensor (TLI4970), relays, and the security controller are also used. Each of these hardware components requires their own driver in the Contiki OS environment. Consequently, these new drivers have to be added into the existing folder structure to "core/dev".

IoT Device Specific OS Requirements

In the actual definition of this thesis, two different basic types of IoT devices exist: a gateway, and several IoT nodes. For each dedicated device is designed a high level application for the Contiki OS. The high level application is responsible to coordinate all defined tasks to operate correctly inside the complete IoT environment. However, the next section is focused on the definitions on the high level applications for the gateway and also for the IoT nodes. The high level application for the nodes is additionally separated into a variant for the smart outlet and for the smart switch.

- **High Level Application for the Gateway**
  The gateway is the central communication device inside the entire constructed IoT environment and additionally acts as border router between the WSN and the WWW technologies. The main tasks of the application are to run a webserver with the support of processing dynamical content, managing several UDP sockets, nodes management, and key management. These applications have to be managed by the gateway and lead to a functional device.

  The implemented webserver should support the visualization of standard HyperText Markup Language (HTML) websites, which can be extended with dynamic content. The support of dynamic content inside the website allows the developer to create an interactive website, where the user has the possibility to interact with the connected IoT devices. One possible solution would be the usage of Asynchronous JavaScript and XML (AJAX) enabled content in the webserver. The advantage of using this feature is to reduce the transmitted data size between the devices and the gateway because only these content data are requested by the devices, which are currently required to be displayed.

  Establishing a communication between the gateway and the IoT devices, two different UDP sockets are required. One UDP socket is configured as a server to handle all incoming connections and the other one as client. The server socket is necessary

to be able to receive information from all other nodes, and the client socket is used to send information back to the IoT nodes.

Figure 3.19 depicts the general state diagram of the various workflows of the gateway. The workflows are separated into two main components: main and webserver. All previously described applications are listed with their functionalities as well as a rough overview of the sequence of called functions.

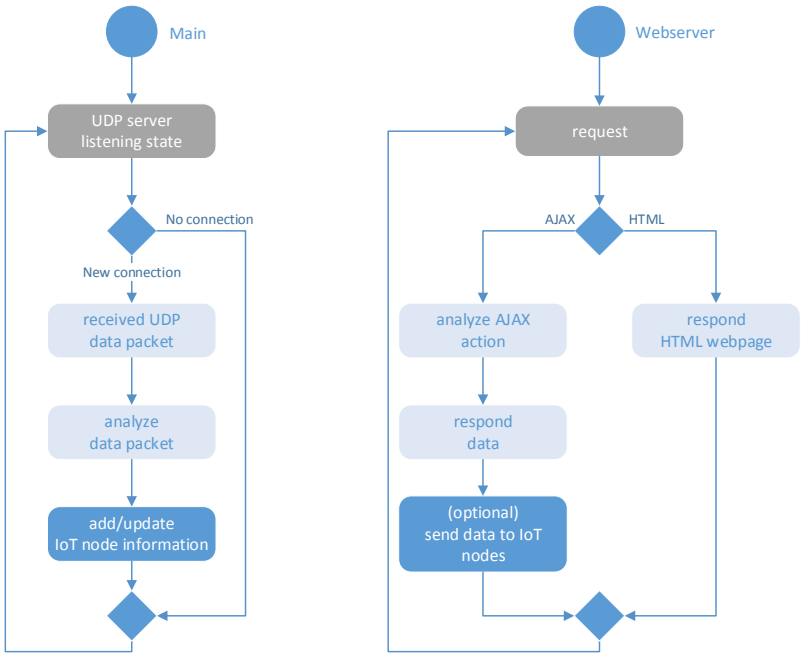


Figure 3.19: General state diagram of the workflows of the gateway. The workflows demonstrate the main process and the webserver.

- **High Level Application for IoT Nodes**
  Each IoT node is basically structured in the same way from the perspective of the application. Each node requires two UDP sockets for complete communication. One UDP socket is used for managing the process of receiving data messages from other devices, and the other socket connection is responsible for sending data back to connected devices inside the IoT environment.

  In the high level application of the node, a process runs for analyzing and controlling the attached sensors and actuators. The responsibilities on the process are to manage their status information and to handle new measured data. Another task of the application is to handle the updating process of node information. The information has to be updated on one hand to the gateway over the wireless communication channel, and on the other hand to the storage of the security controller. The task for sending the information to the gateway is necessary to display all available status information of the devices on the hosted website. The data transmission to the security controller is for updating the information for the NFC-enabled devices.

Figure 3.20 illustrates the general workflow of the main process of the discussed high level application. The workflow is nearly the same for both new developed IoT devices and only varies in the amount of activated sensors or actuators.

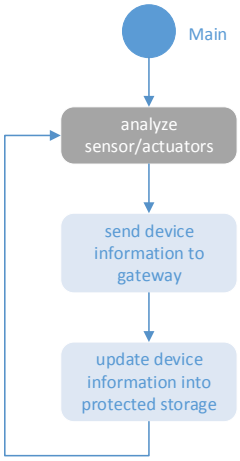


Figure 3.20: State diagram of the basic workflow of a standard IoT node configuration of the smart outlet and smart switch high level application.

Additional Extensions

The standard designed applications for the IoT devices have to be extended with several additional components. These extensions include the definition of the format of the developed communication headers, definition of the file format for storing IoT device specific information, and an API for the security controller.

- **Header Structure for the Wireless Communication**
  In the IoT infrastructure, it is required to define message headers to identify the transmitted data. Each transmitted message, which can either be a status message or a control command, is integrated in the payload field of the general header structure. Table 3.3 summarizes the designed general header message format. In the beginning of the header, a magic number is used to identify the header format from other ones. With the functionality of the magic number, it makes it possible to use different header formats in the same network and to be compatible to older demonstrators. The node type parameter field specifies the type of the implemented functionality of the node, like if it is a gateway, a smart outlet, a smart switch, or something else. The next header fields “sendID” and “receivedID” are used to identify the sender and receivers address, which are used for the identification of used cryptographic keys and for routing the packet to the right recipient. The property field “payloadIsSecured” signals the header format if the included payload is encrypted or in plaintext. With the next field it is possible to define the type of the payload content. Typical content types are: informational messages, control commands, and infrastructure dependent messages. The next property field in this header format describes the length of the attached payload in bytes. Finally, the payload is appended directly after the general message header definition.

Data type	Name	Description
uint16	identity	Magic number to identify correct header format
uint8	nodeType	Defines node type, especially the functionality.
uint16	sendID	Senders ID - Unique address in network.
uint16	receivedID	Receivers ID - Unique address in network.
uint8	payloadIsSecured	Identifies if the attached payload is encrypted.
uint8	payloadType	Defines the type of the payload: information, command, and infrastructure
uint8	payloadLength	Length of attached payload.
—	<payload >	Attached payload buffer .

Table 3.3: Overview of the designed generic message header.

The attached payload has two different configurations plaintext or encrypted, which is due to the variation of the type of included payload. The various versions are listed in Table 3.4. One configuration is named “genericMsgPayload” and represents an unencrypted payload content. This structure contains the number of included sensor values and the measured sensor data. The second configuration of the payload header is for the encrypted payloads. The encrypted payload configuration includes the real payload length of the unencrypted “genericMsgPayload” size in bytes and the general header format for unencrypted payload. This encrypted payload format is then encrypted by the cryptographic algorithm and is attached to the generic message header format as payload.

Data type	Name	Description
genericMsgPayload		
uint8	numOfSensor	Number of contained sensors.
sensorInfo []	sensorInfoPayload	Array of sensor data values.
encryptedMsgPayload		
uint8	payloadLength	Payload length of decrypted message in bytes.
genericMsgPayload	sensorMsgPayload	Sensor data information payload.

Table 3.4: The payload header construction exists in two different versions. One for the insecure transmission and the other one for encrypted payloads.

An extra defined structure is available for the measured sensor and actuator data. This format is named “sensorInfo”, and includes two property fields: type and value. The “type” field defines the type of contained measured data. The “value” field contains the present measured value of the sensor.

Data type	Name	Description
uint8	type	Type of sensor value (e.g. current, voltage, power-consumption, etc.)
uint16	value	Value of sensor data.

Table 3.5: Each sensor value is embedded in such a “sensorInfo” structure.

- **Storage Format of IoT Device Specific Information**
  Each IoT device has the possibility to store device specific information on the security controller. This opportunity enables the transfer of stored information of the IoT device to NFC supported devices. The storage format should be kept very generic to support different variations of IoT devices. A simple storage format for this information is JavaScript Object Notation (JSON). JSON is a lightweight open-standard file format, which is a human-readable text with the possibility to transmit data objects. These data objects can be an attribute-value pair or an array of data types.

  As a consequence, the JSON format is used for storing information. Fixed content information is the “info” field inside the JSON format. This container contains typical information of the IoT device like: address, device type, and secured status. If the IoT device is specified with sensors or actuators, then this information is stored in the “sensors” field as an array. An example of a stored IoT device information in JSON format is illustrated in Figure 3.21. This example includes the fixed content information with three sensor values.

```
{
  "info":
  {
    "id": 62301,
    "type": 1,
    "secured": 1
  },
  "sensors": [
    {"type": 2, "value": 124},
    {"type": 1, "value": 32},
    {"type": 0, "value": 1}
  ]
}
```

Figure 3.21: Example JSON file format for an IoT device with three sensors.

- **API for Security Controller**
  The provided functionalities of the security controller can only be accessed with a special defined API. The API works as translation unit between the software component and the hardware abstraction layer. Each designed method in the cryptographic method has to be added separately to this interface definition. In general, the API is a collection of all accessible functions from the security controller, which manages the hardware components.

3.6.3 Security Controller

The security controller, selected in this master thesis [2], is equipped with a Java Card operating system, which can be extended with self-programmed applets. This feature is essential to build a secured IoT device because the special requirements and methods can be directly programmed into the chip. The OS in the Java Card is designed in a way that the executed operations cannot be revealed through different attacks like side channel attacks or similar. Additionally, the security controller supports hardware acceleration modules for different cryptographic algorithms like AES, ECC, and RSA.

From the security controller, different cryptographic functions are requested to enable a secured environment for the distributed IoT devices. With the aid of the defined requirements, a Java Card applet is designed, which provides an interface to the integrated functions of the controller. The security controller has two different communication channels, NFC and UART, where the commands can be sent to the controller.

Secured Data Storage

The security controller includes an own NVM for storing applet specific information. The developed applet should use the NVM for storing content information of the attached IoT node. The content information contains detailed information about the node like address, secure status, node type, and configured sensor data. The stored information is formatted in the previously define JSON format (see Section 3.6.2)

In addition, all cryptographic keys should be stored in the secured NVM. The amount of stored keys depends on the node type and mode of configuration. Consequently, the storage of keys should be managed in a dynamic concept, in order to provide high flexibility relating to expansion.

Provided Functionalities

The security controller has to perform different working processes for enabling a secured IoT environment. Due to the fact that the used security controller has two communication channels, the available functions have to be separated and classified into internal and external functions. Internal functions are only allowed to be executed for the communication with the trusted microcontroller. This set of functions includes actions like transmitting plaintext information to the security controller, forcing an encryption of provided data, and setting node configurations. External functions have to deal with mobile devices with NFC support and require a higher security level. Due to external usage, all exchanged

[2] Slightly modified internal Infineon test-chip, based on variants described at www.infineon.com/security

information are protected by a secured communication channel between the external devices and the security controller. The communication channel to the microcontroller is also allowed to request the execution of external functions.

In the next section, a list of functions is provided which have to be available in the applet for the designed IoT environment:

- **External and Internal Available Functions**
  - Receive encrypted device content information
  - Evaluate session data
  - Create a new session
  - Transfer pairing data from and to the device
  - Get node type
- **Only Internally Available Functions**
  - Get and set device ID
  - Set device content information
  - Set node type
  - List all paired device IDs

Chapter 4

Implementation

This chapter focuses on the implementation of the overall IoT-Security demonstrator. The development environment, including the software tools is described, followed by explaining implementing the hardware components. Finally, each designed software component of the thesis is presented by their functionality and realization.

4.1 Development

This section provides an overview of the applied workflow and the development environments that were used. The development environments are shortly described in their functionality in order to operate with the chosen hardware platforms and devices.

4.1.1 Workflow

The implementation of this master thesis requires a workflow which combines the developing processes of hardware and software components. In the first stage, it is necessary to develop the hardware components for the various IoT devices inside the environment. Consequently, the hardware components are constructed for the gateway and the various IoT nodes, like smart outlet and smart switch. Next, the implemented hardware components are verified and tested on their correct functionality. If these tests are positive, then the software components can start to be developed.

The first software component, which should be customized, is the embedded OS for the microcontrollers. In order to do this, the "Contiki OS" is enhanced by miscellaneous drivers for the used hardware components. After a basic operative hardware platform with the devices of a gateway and several nodes, the OS is extended by the network protocols and the key management system. In the process of developing the key management system, the implementation of an Android application is also included. This Android application is used among other things for exchanging the cryptographic keys. For this intention, it is necessary to develop a "Java Card OS" applet for the security controller as well.

After these steps, the implementation is focused on the front-end platforms for the user, including a website on the gateways webserver and some additional operative functionalities in the Android application.

The last step of the overall workflow is to test the entire constructed IoT environment on their functionality and operational usability.

4.1.2 Used Firmware/Software Development Environments

The development process requires to use a different software tool for implementation. The variation of miscellaneous tools is introduced by the different hardware components and platforms. The microcontroller series from Infineon, namely XMC, provides an own development platform, which is called DAVE. The security controller is deployed as a Java Card environment, therefore it is required to use a special tool with additional API functionalities to the Java Card OS. In the next section, all various environment tools, which are used in this thesis, are described by their main functionality and their common field of application.

DAVE (for XMC Microcontroller Development)

The development environment DAVE [82] includes an Eclipse [83] platform with useful extensions for graphically configuring the microcontrollers. The possibilities in configuration include the internal hardware modules, pin mapping, pin definition, and also integrating pre-developed applications. These pre-developed applications contain various software modules and high level applications for a rapid developing process, like webserver and API functionalities for SPI, UART, I^2C, and so on.

In general, the DAVE environment is equipped with configuration tools, which support the developer with important information of the used microcontroller. In addition, all available hardware modules are listed for the selected microcontroller which provides a good overview of all functionalities. The integrated source code of the hardware modules are tested and provide verified configurations. The advantage of the support in the configuration process of internal modules of the microcontroller is that the configuration can proceed easily in a GUI without operating deeply with the registers. The GUI also checks the input values of the configurations on plausibility and informs the developer in case of violations regarding configurations and interconnections of the internal modules. Currently, Infineon supports two different versions of Dave, namely 3 and 4. In the implementation phase is commonly used Dave 3 with some extensions of Dave 4.

Java Card OS (for Security Controller Development)

Java Card Integrated Development Environment (IDE) is used for developing applets for the security controller. This IDE is equipped with a full development environment, which enables the developer to compile and upload the finished applet to the Java Card. In addition, this platform can also be used for configuring the Java Card by several properties to fulfill the requirements for a typical operation.

Eclipse (used for Contiki OS Development)

Eclipse [83] is a universal development platform, which supports nearly every programming language. In this development process, Eclipse is used for the composition between embedded OS and the generated configurations settings for the XMC microcontrollers.

Eclipse is used in the version Neon.3 (4.6.3) with standard extensions for a programming environment for C and C++ language. The main development is concentrated in this IDE.

Android Studio (for Smartphone Application Development)

Android Studio [84] is a development environment for constructing Android applications. This environment enables the developer to construct applications for different Android versions and also for miscellaneous devices easily. Android applications are programmed in Java, and the applications are usually organized in activities. Android Studio is used in the version 2.3.3 for the developing process.

4.2 Modified/Redesigned Hardware Components

Various hardware devices to establish a full workable IoT environment are required. A basic IoT network consists of a gateway and several nodes. The following section is focused on the hardware components used in order to create these individual IoT devices.

4.2.1 Gateway PCBs

The gateway, from the hardware device's point of view, requires a lot of different hardware components to meet the defined requirements. Due to the availability of various evaluation boards and kits, it is not necessary for the gateway to develop everything from scratch. The base element of the gateway is given by an evaluation kit from Infineon, which provides a complete development platform with a microcontroller, debugging system, and several hardware components. The functionality of this basic board can be extended by the usage or new development of add-on shields.

Basic Board

The basic board of the gateway is an evaluation kit, which is named XMC4500 Relax Kit. Detailed information about the evaluation kit can be found in the user manual [85]. Figure 4.1 depicts the evaluation kit with a detailed description of its integrated components. The XMC4500 Relax Kit is equipped with a XMC4500 microcontroller as a main component, and is enhanced with several hardware components like micro SD Card, Ethernet module, micro USB interface, and much more. The pin headers on both sides of the board are essential to build add-on shields for extending the base system with additional functionalities. The following add-on shields should extend the base board with a communication system for the IoT environment and a security controller. This procedure of adding additional add-on shields to this evaluation kit are a suitable workflow for the gateway because the gateway does not have the limitations of a small construction area. The reason for taking this approach is because the gateway is located at a static location and has a fixed power supply.

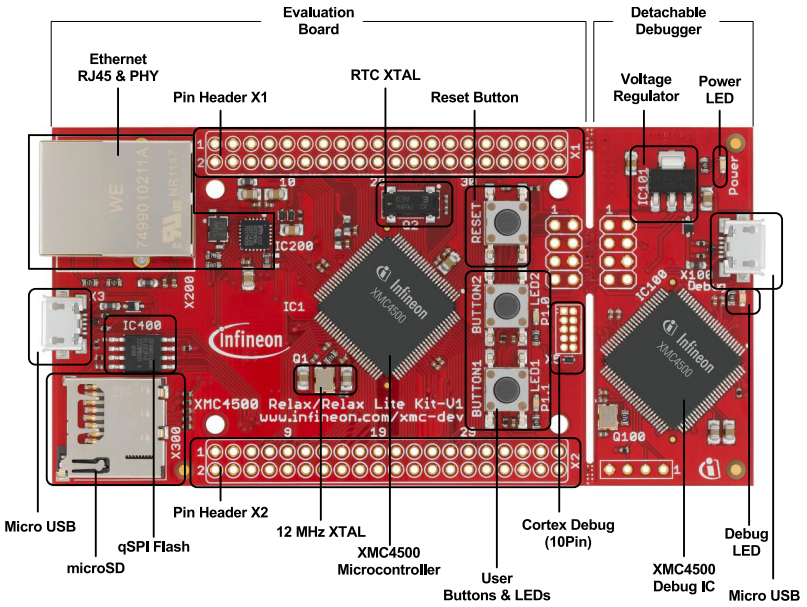


Figure 4.1: Evaluation kit XMC4500 Relax Kit with labels of all necessary components. Illustration taken from [85].

The hardware components are:

- **XMC4500**
  For this gateway, it is necessary that the microcontroller XMC4500 has enough calculation power and flash memory. The XMC4500 microcontroller impresses with the amount of integrated hardware modules, the existence of several input/output pins, and large storage spaces for the developed firmware. These features are necessary in order to fulfill all requested tasks.

- **Ethernet RJ45 and PHY**
  The Ethernet socket (RJ45) and its physical hardware elements enable the connectivity to the WWW with an IPv4 configuration. The maximum transfer rates for this interface is defined by 10/100 Mbit/s.

- **Micro SD Card**
  The purpose for using the micro SD Card is to save ROM storage on the microcontroller. Files of the webserver should be outsourced to the SD Card to be flexible to update the content, and to save storage space on the microcontroller. This card is directly connected to the microcontroller over a special defined bus system.

- **Micro USB**
  The micro USB interface is used to provide a debugging interface for the embedded OS. The integrated USB controller can emulate a Component Object Model (COM) interface. The virtual COM interface is used in order to be able to redirect debugging messages from the embedded OS to the terminal of a computer.

- **Debugger**
  The on board debugger is a useful component to program the compiled software to the microcontroller. Additionally, this component supports the functionality of debugging the microcontroller in complex operating situations.

Add-on Board for IoT Communication

For the communication with the IoT nodes, an add-on shield for the XMC4500 Relax Kit is used, which was developed by Infineon. On this add-on shield is placed the TDA5340 as transceiver module. This module is equipped with an antenna matching network for $868MHz$. In addition to the standard operation unit, a flash storage on the add-on shield is also installed, which will not be used in this thesis. Figure 4.2 illustrates the add-on shield with the TDA5340 for the XMC4500 Relax Kit.

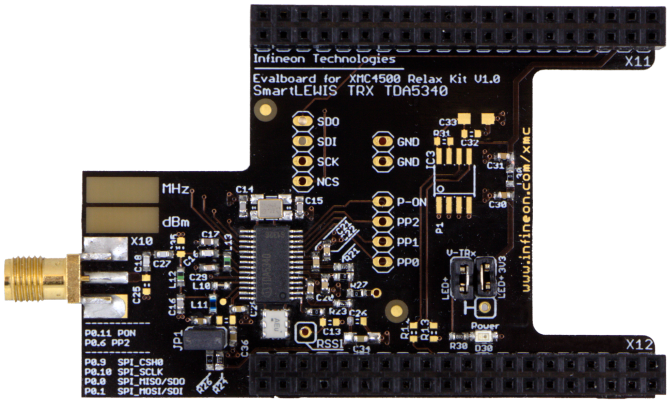


Figure 4.2: Development board for TDA5340 with pin compatibility to the XMC4500 Relax Kit.

Developed Add-on Board with Security Controller

The entire construction, including the evaluation kit with the extension of the TDA5340 add-on shield, is nearly completely constructed. The only functionality which is missed are the components for improving security. Therefore, it is required to design an add-on shield with the security controller and a printed NFC antenna. This add-on shield is developed to fit on the top of the XMC4500 Relax Kit evaluation board and can operate in parallel with the other add-on shields. Figure 4.3 outlines the basic components of the developed extension shield with their used components. The essential parts of this shield are the security controller, which can be switched off with an adequate circuit, and an on-board printed NFC antenna with the possibility to connect an external one to the board.

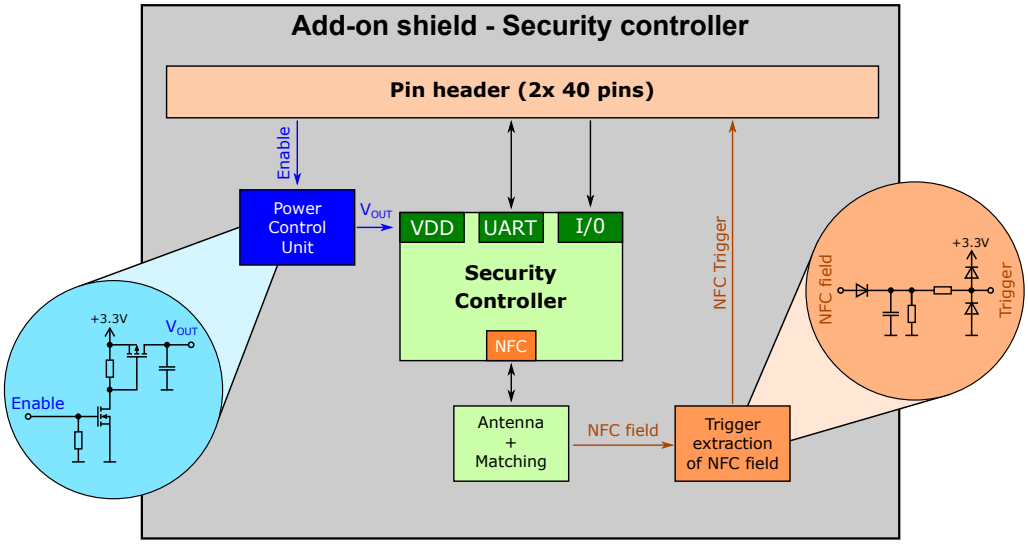


Figure 4.3: System architecture of add-on shield for security controller with integrated NFC antenna.

The power control unit on this board is used to control the power supply from the security controller in order to switch the hardware component on and off. The security controller can either be powered by the external power supply, or by the generated electromagnetic field of a NFC-enabled device. This power control unit consists of one NPN- and one PNP-Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) to build the required circuit. The security controller can be switched on with the power supply by pulling the "Enable" pin to high state.

Additionally, a circuit for generating a trigger signal out of the NFC electromagnetic field is also placed. This trigger signal is used as input for an interrupt control unit, in order to become aware of the process if a NFC-enabled device leaves the communication area of NFC antenna. This process is required because the security controller provides no interrupt signal for the described activity. Therefore, the best way to detect the leaving process is to generate the trigger signal with a diode and a capacitor, as it is depicted in Figure 4.3. In addition, the interrupt input pin of the microcontroller is protected against high voltages. The trigger signal is protected with two Schottky diodes to limit the voltage between $0V$ and $3.3V$ plus the voltage drop of the Schottky diodes.

4.2.2 IoT Node

The nodes in the IoT environment face special limitations like small construction area and low-power consumption. The implementation process of the IoT node construction is divided into a basic board and multiple extension boards with different functionalities. The next section focuses on the implementation process of creating the hardware components for the IoT devices of smart outlets and smart switches.

Redesign of Basic IoT Node Board

The Printed Circuit Board (PCB) concept is in principle based on an earlier WSN demonstration, but is redesigned from scratch due to various required enhancements and replacements of both microcontrollers. The main components of a standard IoT node are a microcontroller with the extensions of a transceiver, and a security controller. With the construction of this basic IoT node board, the node should be able to communicate within the IoT environment and to operate with cryptographic functionalities by the support of the hardware element. Figure 4.4 depicts the complete construction concept of the basic IoT node board. In addition, Figure 4.4 illustrates the interconnectivity and communication types among each hardware component. The central point of this board is the microcontroller, which should be small, but powerful enough to fulfill all tasks in a suitable time.

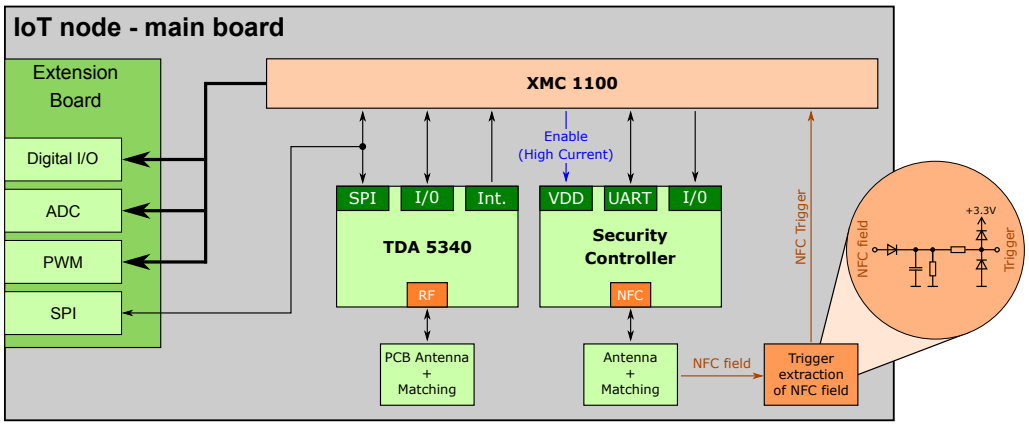


Figure 4.4: Implemented architecture of the basic board for an IoT node.

Detailed information of the interconnectivity between the hardware components are provided in the next paragraphs. All hardware components are discussed, which are illustrated in Figure 4.4. The selection of the right components with their communication interfaces are important in the implementation phase, thus the resources of the microcontroller are used efficiently and results in a high utilization.

- **XMC1100**
  The XMC1100 [74] is a small 4x4mm microcontroller. Nevertheless, the microcontroller provides enough interface possibilities to fulfill all requested tasks. Important modules of the microcontroller are the USIC and ERU unit. The USIC module includes two channels, which can operate independently from each other. The current basic configuration uses both USIC channels: one as UART configuration for the security controller, and the other one is initialized for SPI mode. SPI is used by the transceiver unit in order to communicate inside the IoT environment.

- **TDA5340**
  As previously mentioned, the transceiver is connected to the microcontroller with a SPI interface. In addition, some digital input pins are used at the microcontroller for signaling special notifications. The TDA5340 also generates interruptions for signaling if a new data packet is received by the radio unit.

- **Security Controller**
  The security controller is connected with the microcontroller via an UART interface, which is specified by the ISO7816 standard. After the detection of a NFC-enabled device leaving out of the communication range, the microcontroller is informed by the created trigger signal to check the buffer of the security controller if a new command or parameters are available. The security controller has no dedicated interrupt output pin, which can be used for signaling the leaving process or that new data are available. Therefore, it is necessary to create an own trigger signal out of the NFC field. The circuit for setting up a trigger signal is the same for the gateway and has the same functionality.

- **Extension Board Connectivity**
  The extension board connectivity is a combination of all free input and output pins of the microcontroller. Additional hardware components which are required to use a communication interface, can only use SPI because the XMC1100 does not have a free channel to implement for example I^2C. Among these connectivity options various digital input/output pins, interrupt inputs, ADC channels, and PWM outputs are available.

Extensions of Basic IoT Node

An IoT node with the basic board configuration can communicate inside the environment, but cannot provide useful sensor information or actuator functionality because there are no hardware components available. The purpose of the extension board is to provide these functionalities to the basic board. In this master thesis, two different extensions for a smart home application are constructed. These extensions should provide a good starting point inside a connected home and can simply turn a general home into a smart one.

These two extensions are comprised of a smart outlet and a smart switch. The design process defines the functionality of these two extensions, and also the selection of used hardware components. This next section discusses the process of creating the hardware devices with their major implementation solutions.

- **Smart Outlet**
  The smart outlet supports the basic board with the functionality of switching the power supply inside a typical home. In addition, the node has the possibility to measure the current flow and the attached voltage of the connected device. Consequently, the node is able to calculate the power consumption of the attached device, due to the measured parameters. A complete overview of used hardware components and their interconnections are illustrated in Figure 4.5.

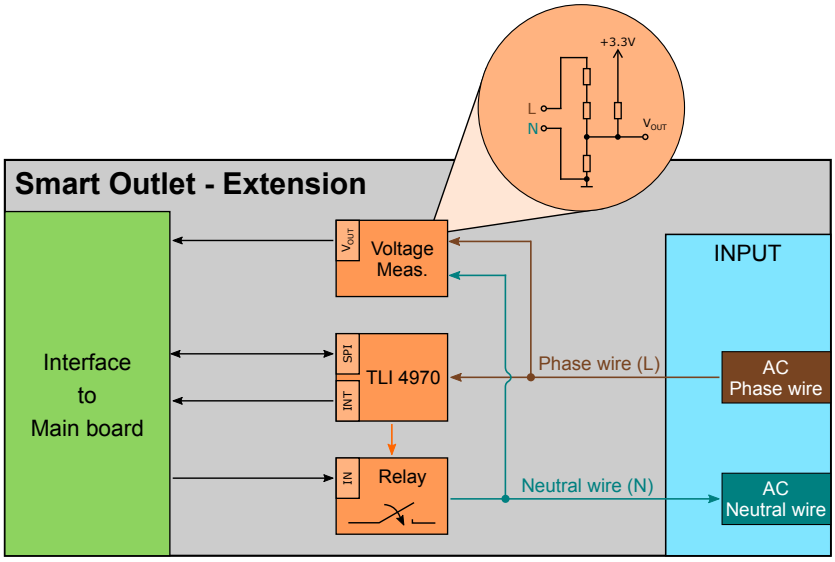


Figure 4.5: Detailed illustration of the smart outlet extension board with the communication interfaces and used components.

These hardware components use various communication interfaces for the interaction with the microcontroller. Due to this fact, it is focused on each hardware element to point out the information, which is necessary in the implementation process.

The designed platform includes following hardware components:

- **Relay**
  For the relay component is used a solid state relay with a zero-cross detection system. This relay is specified for voltages of up to 600V AC and can switch a current of maximum 1.2A. The controlling interface is isolated by the switching unit by a Light Emitting Diode (LED) unit. Consequently, the process of controlling the device is very simple: by powering the LED unit. The used relay has no safety functions, which are able to detect an over current or other destroying factors.

- **Current Sensor**
  For the task of measuring the current flow of the attached device, it is preferable to use a hall element, ensuring an isolated measuring. The TLI4970 is constructed for this use case, and also provides useful functionalities like over current detection. The measurement values of this element can be received over the integrated SPI interface. The TLI4970 is used with the extension labeling D025T4, which defines the maximum measurable current. In this case, it corresponds to 25A in the positive and negative direction. Equation 4.1 demonstrates the calculation of the corresponding current with the received value ($out[LSB_D]$), which is measured by the TLI4970. The smallest current

change, which could be tracked, is approximately 6mA.

$$I_{out}[A] = \frac{out[LSB_D] - 4096[LSB_D]}{160[\frac{LSB_D}{A}]} \tag{4.1}$$

– **Voltage Sensor**
In order to precisely calculate the correct power consumption of the attached device, it is also required to measure the voltage of the power supply. The connected power supply is an AC voltage with a normally Root Mean Square (RMS) value of 230V in Europe. This RMS value can varying through different properties, therefore it is necessary to measure it with an adequate circuit. On the hardware platform has to be knowingly activated this functionality by connecting two hardware pins due to the non-isolated measurement method. In the standard application is disabled this part in this thesis and is only prepared for the usage, but if there is enough time then this feature will be activated.

The True-RMS value is calculated in Equation 4.2. This Equation calculates the square root of the mean of the square of all measured voltages from the voltage waveform during a specified period of time.

$$V_{RMS} = \sqrt{\frac{\sum_{i=0}^{n}(V_n^2)}{n}} \tag{4.2}$$

In order to get the voltage values of the attached power supply, it is necessary to prepare the measured voltage for the ADC input pin for the microcontroller. The input pin of a microcontroller can only measure values between zero and the reference voltage, which corresponds with the power supply voltage of the microcontroller. Due to the AC voltage supply, it is required to reduce the power level to the input limitations, followed by increasing the input voltage with a Direct Current (DC) offset value because the input AC waveform also includes negative values. The new resulting output voltage after the added DC offset, is always above zero volts. The created circuit for this task is depicted in Figure 4.5 in the detailed illustration of the voltage measurement block.

- **Smart Switch**
The functionality of a smart switch is to react on pressing a button on the switch. The IoT device tracks the push action and forwards the trigger event to the microcontroller, which subsequently sends specified commands to another paired IoT node. The hardware construction of this extension board is very simple and only requires one input pin. This input to the microcontroller is performed as digital input pin with ideally an external interrupt support. The board construction is illustrated in Figure 4.6 for a better visualization. A common way to debounce the button switch is to use a combination of hardware and software based solutions.

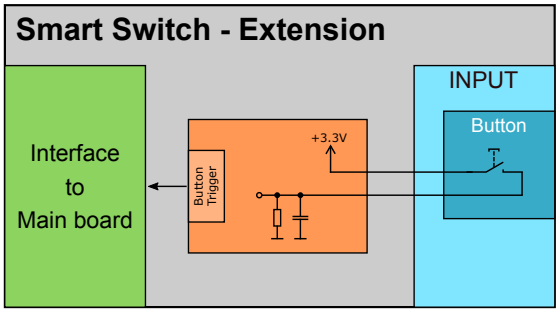


Figure 4.6: Detailed overview of the used components and connectivity of the smart switch extension board.

Trade-offs regarding PCB Form-Factor Variants

In the process of designing the hardware PCB of the IoT nodes, it became apparent that the separation into a main board and several extension boards is not simple to solve, due to the limitation of the minimal construction height. The restriction of the limited height of the entire hardware structure is limited by the installation space, because behind a standard wall outlet there is only a small place. This small area behind the outlet has to be used in an economical way. In addition, the difference on hardware components on a smart outlet and a smart switch is extremely low. The smart switch requires only an extra button input. In conclusion, a hardware platform is designed on a tiny PCB, which can be used for both device configurations.

4.3 Modified/Redesigned Software Components

A big part of the software components is the embedded OS for the various IoT devices. Extra applications are required to be developed for miscellaneous parts of the project. These parts are the applet for the security controller, a website for the gateway, and an Android application to control and force the key exchanges.

4.3.1 Contiki OS

Contiki OS is an embedded OS with a 6LoWPAN stack for the communication inside the WSN part of the entire IoT infrastructure. The base functionality of the embedded OS provides the communication stack protocol and basic operational methods. Therefore, this OS can be used for the developed hardware platforms as a base system, which has to be modified with various functions. Some functionalities and basic configurations are already developed by Infineon, which includes the driver components for the TDA5340, network stack configuration, and basic process definition. The next section discusses the processes of modification of the enhanced Contiki OS. The modification is necessary to be able to operate with the designed hardware devices. In addition, the focus is lead on the different available applications including the list of running process threads.

The workflow for the adoption of the Contiki OS is divided into several intermediate stages. In the first step of adopting the OS, the new designed hardware platforms are required

to be added to the project structure of Contiki OS. After this step, the communication platform including the protocol stack 6LoWPAN, can be configured and tested in their functionality. Once this is implemented and runs successfully on the hardware platforms, it should be possible to send simple data packages between the devices. After implementing these configurations in the OS, different high level applications can be developed for the IoT devices.

CPU Extension

The new designed hardware platforms use microcontrollers, which are not officially supported by Contiki OS. Consequently, the microcontrollers from these new hardware platforms have to be added into the existing structure. The basic configuration and header files of the microcontroller can be extracted directly from Infineon. The only additional adaptation on these configuration files have to be processed for the integration into the API compatible interface of the OS. The OS provides several API functionalities for initialization and using low level configurations of the microcontroller. Due to this reason, the Infineon files have to be adapted in the integration process to the existing interface. In the design process, two microcontrollers with a central core processing unit from ARM are selected. The new microcontroller files are placed under the folder structure "cpu > arm > xmcXXXX". The configuration of the microcontroller includes the configuration of the base system of the CPU. The base system consists of modules like the internal clock generation, start-up scripts, and register definitions.

Platform Configurations

It is possible to define various platforms in the Contiki OS environment. A platform configuration is defined as one hardware device with several external attached components. Ideally, the platforms should be able to be independently exchanged to the applications. Therefore, the hardware platform can be selected in the Makefile before the complete project is compiled.

In this master thesis, two new hardware platforms are created, which have to be integrated to the Contiki OS environment. These two hardware platforms are named in the Contiki environment "xmc4500-SecureGateway" and "xmc1100-IoTNode".

- **xmc4500-SecureGateway**
  The platform of the secured gateway requires to define and to configure all external used components, which are on the basic board and on the add-on shields. The configuration of the entire platform is a little bit complex, due to the high amount of various external components. In order to simplify this configuration process, the Infineon development environment DAVE is used, to configure the internal modules of the microcontroller. This tool enables the developer to configure the internal modules of the microcontrollers in a simple way through the provided GUI. After the complete configuration process in the DAVE IDE, the created configuration files are integrated into the Contiki OS structure. In addition to the configured internal modules, all pins of the microcontroller are configured with the requested functionality. In conclusion, the platform configuration of the secured gateway provides

configurations information for following modules: Ethernet, SD Card, USB, UART, SPI, PWM, and ERU.

- **xmc1100-IoTNode**
  The platform "xmc1100-IoTNode" is designed for the IoT node configuration. In this platform, all pins of the microcontroller are configured, which are used in the environment. Furthermore, all internal modules of the microcontroller are configured to operate with all external devices. These modules are the watchdog timer, ERU, SPI, UART, and PWM. The process of configuration is also performed in the first stage in the DAVE IDE and is subsequently added to the Contiki OS file structure.

Network Stack Configuration

The network stack has to be configured in the Contiki OS in a way to operate in the defined IoT environment. The configuration is based on previous Contiki OS-based demonstrator in order to stay compatible. The configuration process of the network stack includes different layers, which starts at the top with the 6LoWPAN and ends with configuring the hardware component of the transceiver module. Figure 4.7 depicts the network stack in a clear overview structure with its basic configuration settings. In addition, this Figure illustrates that some drivers have included the word "null" in the naming. This special naming means that the protocol for this part of the network architecture is disabled. The reason for the disabling of the additional protocol is due to the used transceiver module. The transceiver module can transmit in the current configuration only 288 bits at one time. In detail, this means if more protocols are enabled, then the length of the headers are increased, and consequently the resulting payload is reduced with every additional enabled protocol standard. Therefore, in this configuration only the essential protocols are enabled. If there is enough time at the end of this thesis, then the driver for the TDA5340 can be extended to be able to send bigger payloads by filling the buffer during transmission. In the transportation layer are activated UDP and ICMP protocols, in order to send information to other participants. An essential configuration for the IoT architecture is the network layer, which uses 6LoWPAN as communication protocol.

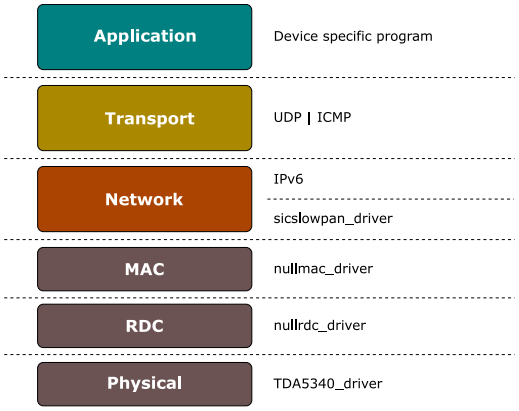


Figure 4.7: Network stack configuration in Contiki OS for IoT environment.

Modified Gateway Application

In principle, the gateway works as border router between the WWW and the WSN. In order to process the task as border router, the gateway requires the support of different features, for example:

1. UDP connections to handle the communication inside the WSN
2. Stored node information for the webserver
3. Host webserver with support of dynamic content
4. Basic key management system
5. Some general features to fulfill all requirements (e.g.: Node management, interrupt handling, neighbor discovering)

Several processes are required for these features which perform all described tasks. These operations are split into several processes, which are scheduled by the OS. The following list describes each process by their functionality and common responsibilities.

- **udp_server_process**
  Figure 4.8 depicts the flow chart of the UDP server process, which is slightly modified to the original based one. The UDP server process is responsible for managing the communication between the IoT nodes and the gateway. For this task the gateway establishes a UDP server socket as a listener, which waits for incoming connections. After the connection is established, the gateway has the opportunity to receive incoming data packages by any IoT node. In general, input data packages contains IoT node information, control commands, and similar.

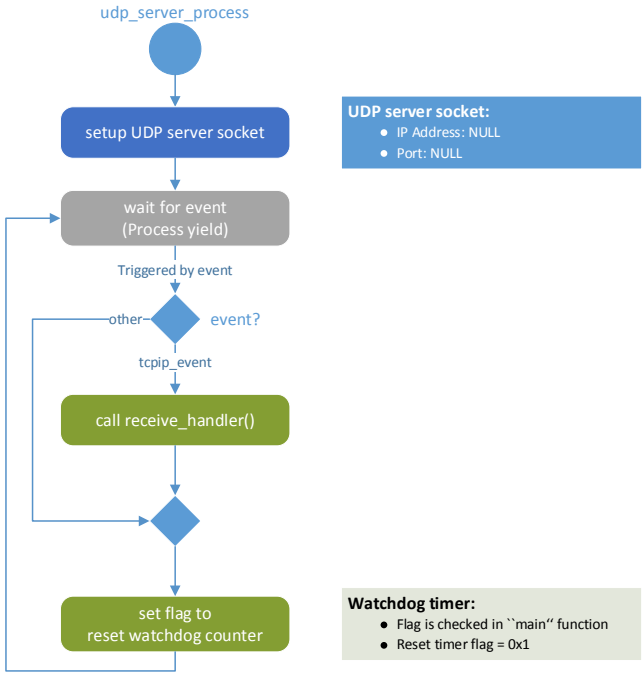


Figure 4.8: Flow chart illustrates the processed tasks of the UDP server process.

- **extern_interrupt_process**
  The external generated trigger signal from the NFC field is observed with an interrupt input of the microcontroller. The interrupt is activated, when a falling edge on the input signal is detected. Due to physical and technical reasons, the input trigger signal can be overlapped with different signals. Consequently, the input signal is debounced to reduce false triggered interrupt signals. In the first step, the input signal is debounced in software by checking at several time positions if the input signal is still low. If the signal is low for the complete testing window, then the OS fires an interrupt signal inside the OS, leading to execute defined handlers. These bouncing effects on the NFC trigger signal are also generated by the NFC communication technology. The processing of the external interrupt process is illustrated in Figure 4.9. In the interrupt routine of the debounced signal, the security controller checks, if a new command or parameters are available in the input buffer. This input buffer can be manipulated by the developed Android application.

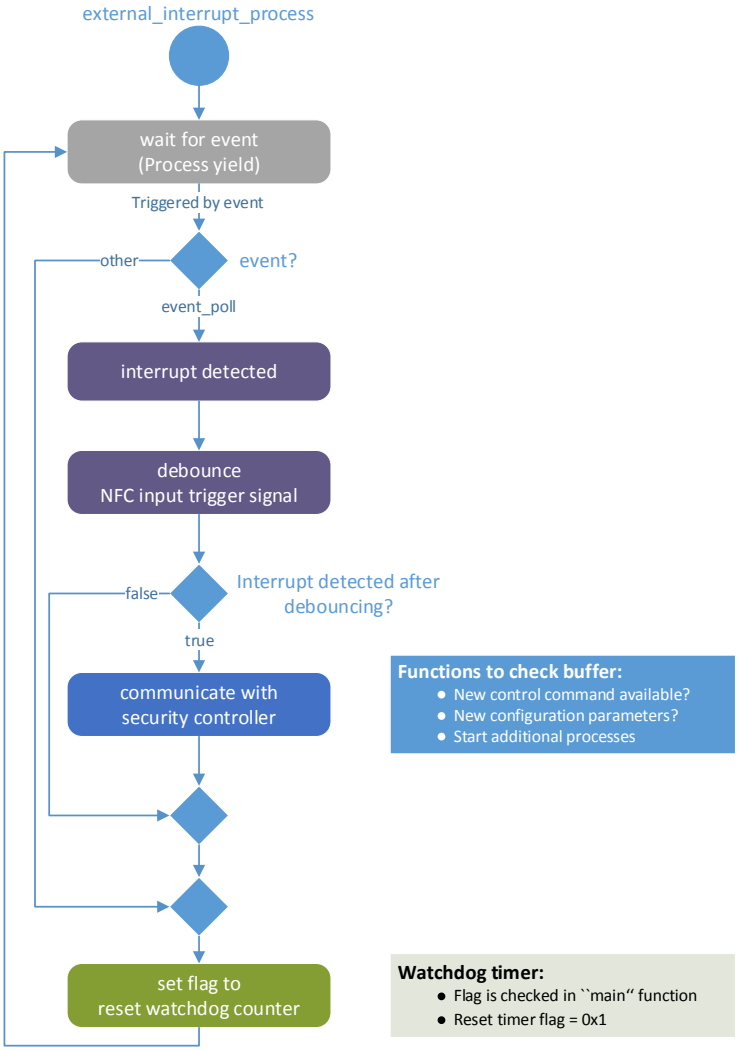


Figure 4.9: The external interrupt process is responsible for the NFC trigger signal. In this Figure is illustrated the workflow of all called tasks.

- **common_process**
  The "common_process" is constructed to control general functionalities in the application. Figure 4.10 depicts the various processed tasks of the common process. The "common_process" is scheduled by the Contiki OS every second. At each call of the process, the output LED is toggled. This output LED indicates that the OS is still in running mode. Furthermore, the process is responsible to write the present device information to the storage of the security controller every 10 seconds. In addition, at every process call the TCP stack is reconfigured for the integrated webserver. The webserver acts as an interface for the border router and establishes an IPv4 connectivity to the WWW environment. In the beginning of the process, the status of the gateway with several parameters is also initialized. The values for the timing specifications can be modified in the header definition of the high level application.

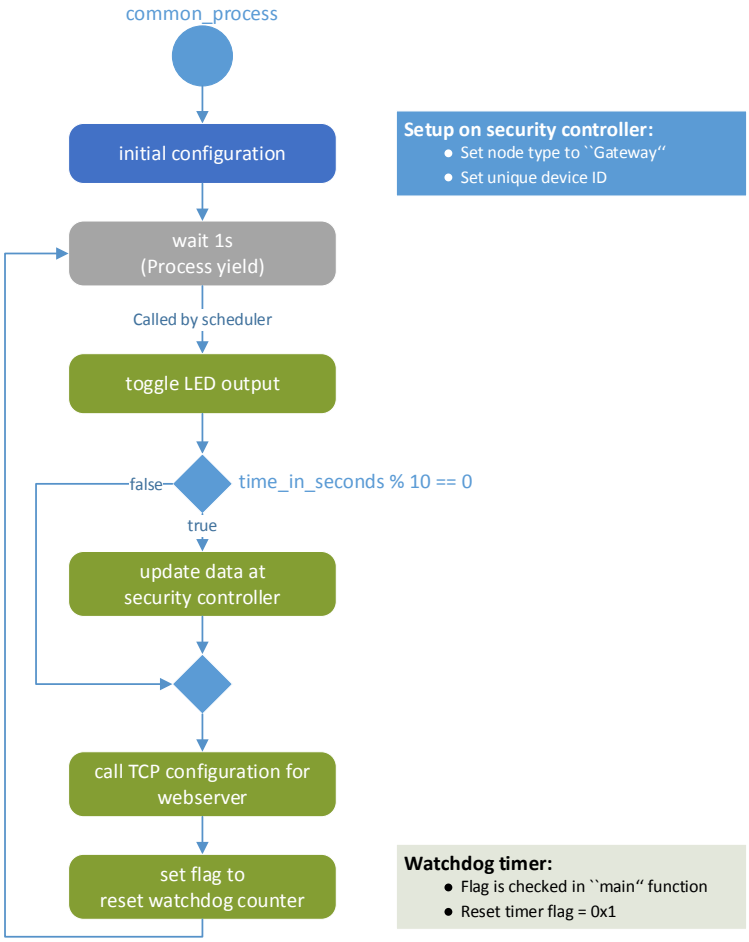


Figure 4.10: The flow chart demonstrates the functionality of the common process.

- **udp_send_process**
  The UDP send process is used at the gateway to respond or send messages to the IoT nodes inside the WSN environment. This process is constructed with the possibility to poll the process from other actions inside the OS. This functionality of polling is required to be integrated inside the webserver, in order to send control commands

to the nodes. The user should be provided with the functionality to control and change the parameters of the selected nodes by pressing buttons on the website. The function of sending packets to the receiver is only processed when a polling request is set for this process. In general, this process is used to transmit control commands and status messages over the established UDP channel inside the UDP server process. In the "send_handler" function, a packet is constructed in the defined architecture with the provided payload, and is subsequently forwarded to the radio unit to be processed for transmission.

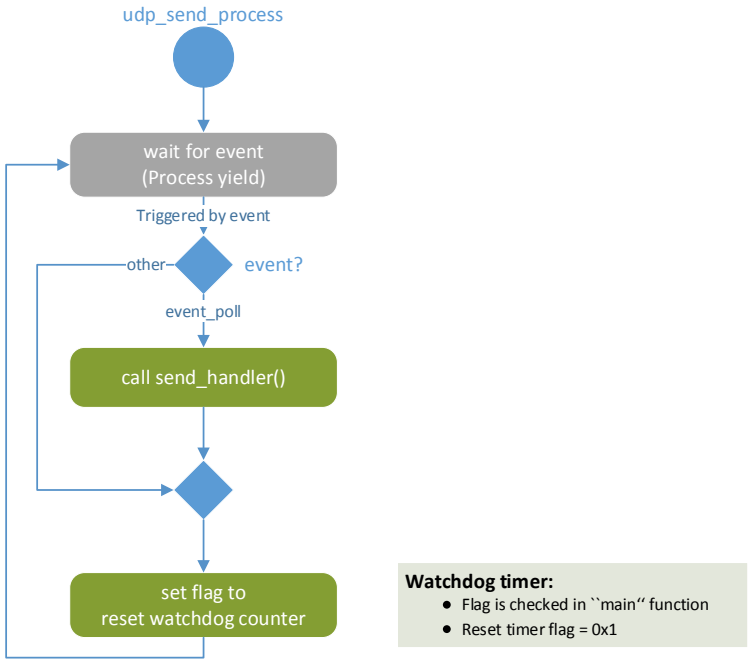


Figure 4.11: Flow chart depicts the actions of the UDP send process.

The sequence of called processes is illustrated in Figure 4.12. In this figure, it is apparent that two different fields of processes exist. The bottom area defines the processes, which are constantly called upon in a pre-defined time schedule. In this definition, the "common_process" is included. This process should be scheduled by the OS every second. The second area in the illustration is the scheduling of the interrupt driven process. All three listed processes are only called when an interrupt arrives at the OS. The external interrupt process is triggered by an external interrupt pin from the microcontroller. The other two interrupts are generated by the UDP connection, for example when a new packet is received or when packets have to be transmitted over the WSN communication channel.

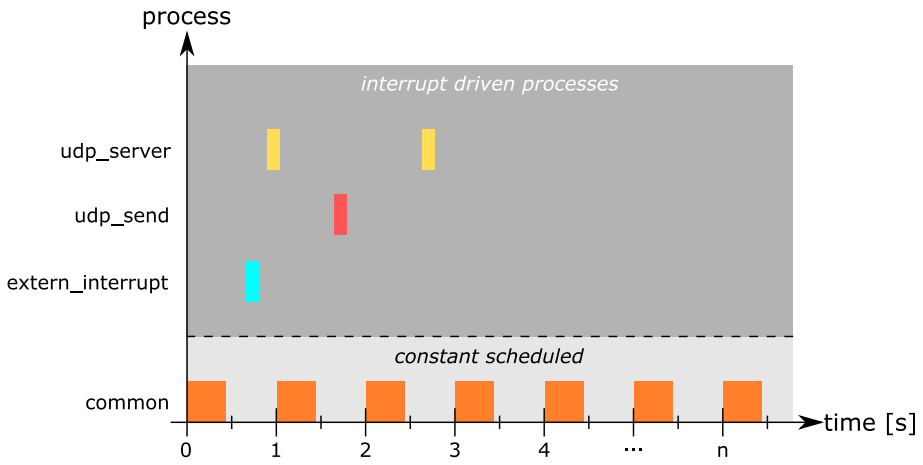


Figure 4.12: Scheduling scheme for all established processes on the gateway.

In addition to the discussed processes, the application is responsible for several sub applications like a webserver, node management, and key management. The next section is focused on these sub application.

Implemented sub applications for the gateway are:

- **Modified Webserver**
  The base application of the webserver is created and configured in the Infineon development tool, DAVE. After the configuration of the webserver in the provided tool it is exported and integrated into the Contiki OS environment. The general functionality of the webserver is to provide opportunities to access and transmit contents of the website. The content for the websites are stored on the SD Card. Consequently, the webserver is also responsible to access the data on the SD Card storage space. The webserver support website formats of "htm" and "stm". Furthermore, the webserver should allow to process dynamical content for the loaded websites. This operation uses the functionality of "cgi" to respond to dynamical requests.

  Three different interfaces are provided to request dynamical parts of the specified content information inside the dynamical part of the webserver. These interfaces are defined and described as follows:

  1. **mote_info.cgi**
     The execution of this function provides a list of all available IoT nodes in the complete IoT environment. The transmitted data is constructed as a string with the node information of address, type, and security status. Each of these parameters are separated by a "%" symbol. Furthermore, nodes are separated in the string by a "$" symbol.
  2. **mote_data.cgi**
     The mote_data function returns, as dynamical content, the detailed information of a selected IoT node. It is defined in the requested header which node information has to respond by the address value. The node information is combined

together to one string. Each parameter of the node is a structure of two values: first value defines the type of value, and the second defines the corresponding value. Each parameter structure is separated in the string by a "$" symbol.

3. **mote_action.cgi**
   With this functionality, it is possible for the website to send control commands to the desired IoT nodes. Therefore, it is possible for the user to control the status of different IoT nodes by pressing buttons on the website. The API for the control commands is defined by an easy structure, which only defines the type of command and the command value. After the processing of the control command by the webserver, the command is subsequently transmitted to the related IoT node.

- **Modified Node Management**
  The node management is responsible to add and remove nodes to the internal structure of the gateway. If node data information is received by the gateway, then it will be added to the management structure. If it is already available, then the parameter fields are updated. The node structure contains a special field, which is called "update". This field is updated at every successful processed information data packet from the nodes. If the information of the node is not updated for several minutes, then the node will be removed from this structure.

- **Developed Key Management**
  The key management system is responsible to manage all pairing information of all integrated nodes. In the first version of the key management system, the sequence of operations is kept, and is simple and supports the basic functionalities. It is responsible for providing a mechanism to renew already distributed cryptographic keys between two paired participants. The process of renewing the current keys can be forced by several opportunities, for example: after some amount of transmitted encrypted messages, after a specified time, or by the user itself. The key management system is also supported by the developed Android application, which handles the key exchange for the first time, and enables a secured initialization of the IoT environment.

Application for Smart Switch and Smart Outlet Devices

The application for the IoT nodes, especially for the smart switch and the smart outlet, is constructed to read the connected sensors and to transmit informational content of the nodes to the gateway. Furthermore, the application is responsible to manage the basic tasks and the basic key management operations. The application is split into several processes in order to proceed with all requested operations. All processes are scheduled by the Contiki OS when they are required.

An overview of the processes, including their functionality, is discussed below:

- **udp_server_process**
  The UDP server process for the IoT nodes has the same workflow than for the gateway. This process establishes a UDP server socket, and manages receiving activities of new packets. The server connection is requested to be able to receive messages from the gateway side.

- **extern_interrupt_process**
  The external interrupt process in Contiki OS is also constructed in the same way as the gateway. If an interrupt is recognized after software debouncing, then the security controller is checked if new control commands or parameters are available in the buffer to be transmitted to the microcontroller.

- **sensors_process**
  Figure 4.13 illustrates the common workflow of the sensor process. The main tasks of this process are to initialize the used sensors on the hardware platforms, and to refresh the sensor values. If a sensor value changed, then a special flag is set in the sensor structure, signaling to the OS that a new sensor value is available.

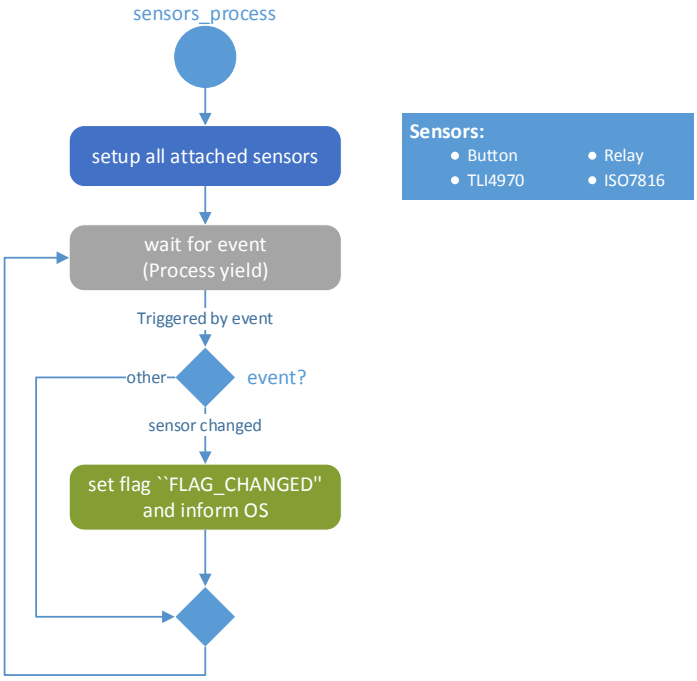


Figure 4.13: Flow chart demonstrates the sequence of the sensor process for this application.

- **udp_client_process**
  The UDP client process is the main process of the constructed application. This process is responsible for initializing the IoT nodes to their desired functionality and to process the common operational tasks. In the first stage of this process, a UDP socket connection to the gateway station is established. This connection

is required to send node information to the gateway, and to also receive necessary control commands. Furthermore, the node is initialized with its unique device ID and node configuration data. After the initialization tasks are performed, the process yields each time and is called by the scheduler approximately every second. If the IoT node is constructed to operate as a smart switch, then the input button is checked if an event is triggered. Otherwise, if it is a smart outlet, then the sensors should be triggered to measure all available values. The next process is to update the node information to the storage of the security controller after every 10 seconds. After every 15 seconds, the node should send a status information message to the gateway to show its availability, and to update the measured information on the website. These timings are modified during runtime to save energy consumption and to provide in the case of an active usage a faster updating process for the website. Figure 4.14 depicts the described workflow of the UDP client process.

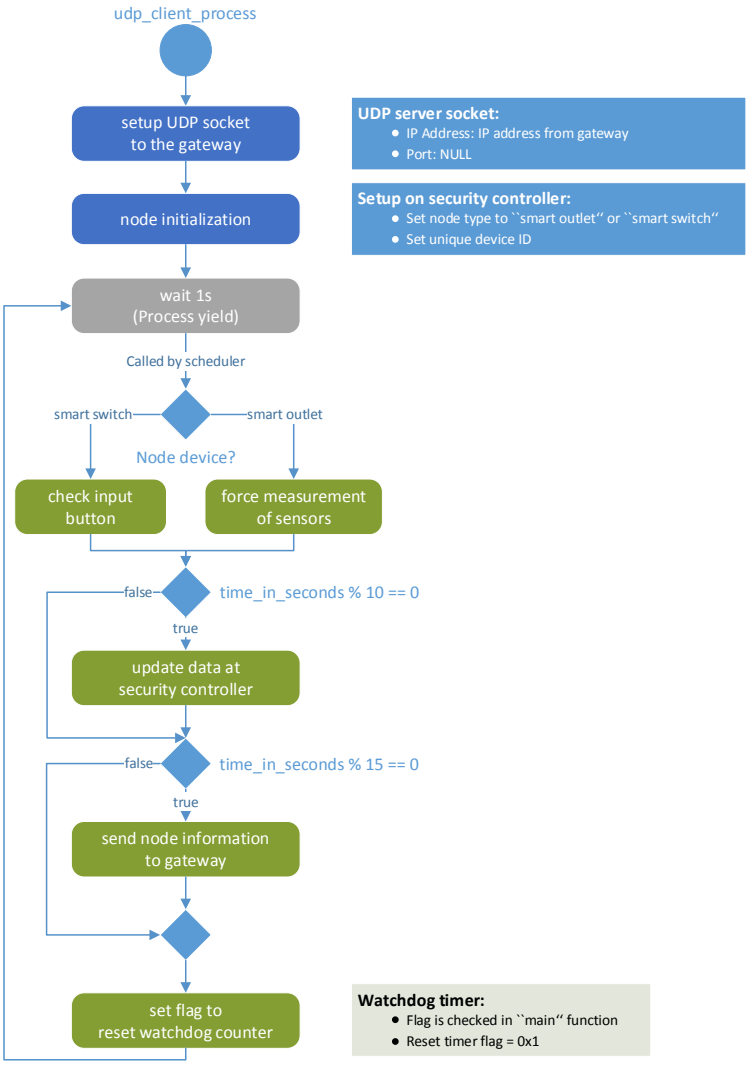


Figure 4.14: This flow chart illustrates the operational tasks of the UDP client process.

The defined processes are supported by two sub applications. These applications are responsible for managing the sensors and the key management. Each of these applications are described below:

- **Sensor Management**
  The sensor management system controls the process of collecting all measurement values of the connected sensors. Under the collection of sensors is the current sensor (TLI4970), security controller, relay unit (AQH3213A), and the input button. The button input is only required by the smart switch platform configuration.

- **Basic Key Management**
  The basic key management system has the responsibility to manage the paired node information inside the security controller and to keep them up to date. Most of the key management system is placed directly inside the security controller. This part of the application manages the storage for the cryptographic keys and protects them against external attackers. The application on the node side covers the process of managing the workflow of renewing and exchanging of additional cryptographic information.

4.3.2 Designed Website for Dynamic Node and Security Management

One of the main front-end platforms of this thesis is the webserver with the provided website. The website is used to visualize all received sensor data and node information of the IoT environment. Other functionality besides the visualization is to send control commands to the available IoT nodes. Each IoT node is equipped with different set of control commands. These control commands include switching a relay or pushing a button.

The complete website consists of three sites: home, dashboard, and information. The sites, home and dashboard, provides the visualization of the content information of connected IoT nodes. In order to reduce the amount of transmission data between the gateway and the end point participant, the website is constructed in a dynamic way for updating only the content, which is required at this time. The webserver on the microcontroller only provides support for a HyperText Transfer Protocol (HTTP) webserver with post request. On the client side, AJAX is used for realizing the dynamic content transmission that is established on post requests.

The next section describes the various sites which are available on the website, focusing on the implemented functionality and on the design process.

- **Home**
  The home site, which is also the starting site, provides the user with a general overview of all connected IoT nodes in the defined environment. On this site, all connected IoT nodes are in a list on the left side. The content information of the selected node is placed on the right side of the site. If one node on the left side is selected, then the website starts a process for periodically requesting the information of the selected node. The dynamic content of the IoT node is requested and replayed by AJAX functionality. The list of available IoT nodes is periodically requested by the website every two seconds. The webserver responds with a list of nodes, which contain the node address, configuration type, and if the communication is secured.

The detailed node information content is only requested by the website when the node is selected in the list. The content information is refreshed every five seconds.

- **Dashboard**
  The dashboard visualizes the same content information than the home site but only in another view. The design of the dashboard is kept simple in tiles. The main difference between the home and the dashboard view is that on the dashboard the user can see all node information at the same time. This site also provides the possibility to control the nodes with their provided functionality. The content data is requested as well with AJAX commands for the website.

- **Information**
  The website, information, provides the user with a general overview of the IoT environment and their used components. This site is only used to provide informative information to the user.

Implemented AJAX Commands

The website is integrated with AJAX request commands, in order to receive dynamic content from the webserver. Therefore, the webserver provides following API functionalities, which are summarized in Table 4.1. On the website as previously discussed, the list of nodes which is updated every two seconds, and uses the "mote_info" functionality to receive the requested content. If a node is selected on the website, then the corresponding content information of the node is requested every five seconds with the function "mote_data".

Request address	Type	Data	Response
mote_info	GET		List of nodes
mote_data	GET	address=<nodeID>	Node information
mote_action	GET	id=<nodeID>&action=<action>& value=<value>	No response

Table 4.1: AJAX commands for enabling a dynamic exchange of content data.

4.3.3 Security Controller - Applet Development

The security controller runs a standard Java Card environment with no pre-loaded applets. The primary security element in this master thesis is the security controller, which provides hardware accelerated calculations of various cryptographic methods for the platforms of diverse devices. The aim for this IoT environment is to secure the connections between the devices with a certain security level and to store private and sensitive information on a secured NVM. Due to these requests, a new applet is developed to resolve all these functions.

The Java Card environment uses as standard interface, the ISO7816, as a communication channel and provides the possibility to use NFC technology. Considering the Java Card environment and the provided interfaces, the commands including the payloads are transmitted in the construction of APDU packets. The designed applet is responsible to receive these APDU packets and to respond with the desired parameters.

As previously mentioned, the security controller is made up of two interfaces, which uses the same communication protocol. During runtime, only one of those can be used, either the UART or the NFC interface. With the process of controlling the power supply of the security controller, it is possible to enable or disable the opportunity of using the NFC interface. The implementation of a secured applet for the security controller requires different functions to enable security enhancements which have to be provided as interface to the outside of the Java Card. These functions are separated into two security levels: internal and external. If functions are declared to be classified as internal, then sometimes they send data in a decrypted state and also provide detailed information of the IoT device. Internal functions should consequently only be available for the connected microcontroller via the UART interface. The external functions, which are protected by several cryptographic mechanisms, are available on both interfaces.

Overview of Implemented Functions

The implementation of the applet waits until an APDU command is received from the communication interfaces, either from UART or NFC. In the first stage, the received APDU command is analyzed by their supplied properties. After this step, the correct function is selected by the state machine. All implemented functions with their required parameters are summarized in Table 4.2. The detailed description of the transmitted data value and return value are not included in this table; each implemented function with their purposed and detailed transmitted content information will be separately described.

Internal						
Function Name	**INS**	**P1**	**P2**	**LC**	**Data**	**Return value**
GET_RANDOM	0xA0	0x00	0x00	length of data		yes
GET_DEVICE_ID	0xA1	0x00	0x00	0x00		yes
SET_DEVICE_ID	0xA2	0x00	0x00	0x02	yes	
SET_NODE_TYPE	0xA3	0x00	0x00	0x01	yes	
SET_DEVICE_DATA	0xA4	0x00	0x00	length of data	yes	
GET_PAIRED_IDS	0xA5	0x00	0x00	0x00		yes
IS_ID_PAIRED	0xA6	upper byte ID	lower byte ID	0x00		
GET_NEW_CMD_ACTION	0xA7	0x00	0x00	0x00		yes
AES_DEC_WITH_AES_VERIFY	0xA8	upper byte ID	lower byte ID	length of data	yes	yes
AES_ENC_WITH_AES_SIGN	0xA9	upper byte ID	lower byte ID	length of data	yes	yes

External						
Function Name	**INS**	**P1**	**P2**	**LC**	**Data**	**Return value**
SET_INTERNAL_ACCESS	0xB0	0x00	0x00	length of data	yes	
EVAL_SESSION	0xC0	0x00	0x00	length of data	yes	yes
CREATE_NEW_SESSION	0xC1	0x00	0x00	length of data	yes	yes
GET_LINKLAYER_KEY	0xC2	0x00	0x00	0x00		yes
GET_NEW_PAIRING_DATA	0xC3	upper byte ID	lower byte ID	length of data	yes/no	yes
SET_NEW_PAIRING_DATA	0xC4	upper byte ID	lower byte ID	length of data	yes	
GET_ENC_DEVICE_DATA	0xC5	0x00	0x00	0x00		yes
SET_NEW_CMD_ACTION	0xC6	0x00	0x00	length of data	yes	
GET_NODE_TYPE	0xC7	0x00	0x00	0x00		yes

Table 4.2: Complete APDU command structure.

Detailed Description of Internal Available Functions of the Applet

Internal functions are not secured in all perspectives, due to the performance reasons of the microcontroller. In order to protect the internal functions of misuse from external users, it requires to enable the internal function set by a secret information. The access to

the internal functions can be gained by the execution of the "SET_INTERNAL_ACCESS" method, which requires as input value a shared secret. This method of gaining the access to the internal functions is only very simple and should be improved to have a stronger security level. This differentiation is required due to the fact that the two interfaces cannot be separated in their accessibility.

- **GET_RANDOM**
  This function requests the security controller to return a specified amount of random generated bytes. The amount of generated bytes is defined with the LC property field of the APDU command. The random bytes are generated by a TRNG module.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** generated random bytes

- **GET_DEVICE_ID**
  This function requests the security controller to return the stored device ID. The return value of a successful execution of this command is an array of two bytes, which are responding to the stored device ID.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** device ID (2 bytes)

- **SET_DEVICE_ID**
  The SET_DEVICE_ID function is used to store the current device ID to the security controller. In the APDU command, the new device ID has to be included. This function is not returning any value except for the trailer.

  Provided and returned data:

  - **Input data:** new device ID (2 bytes)
  - **Return value:** no return value

- **SET_NODE_TYPE**
  The SET_NODE_TYPE function is necessary to define the device type. The microcontroller can define the behavior of the complete IoT device inside the security controller with this function because the applet is slightly different for different node types. Two different configuration types are defined: The value "1" represents a gateway and "2" signals a sensor node.

  Provided and returned data:

  - **Input data:** node type value (0 = default, 1 = gateway, 2 = node)
  - **Return value:** no return value

- **SET_DEVICE_DATA**
  The content of the node data can be updated or set with this function. This process is required that the user has the possibility to request the node data with a NFC-enabled device. The content data of an IoT node includes device specific information and the current sensor values.

  Provided and returned data:

  - **Input data:** Information of the device in byte format
  - **Return value:** no return value

- **GET_PAIRED_IDS**
  This function returns a list of successfully paired devices. A paired device can represent a node or a gateway where all cryptographic information is available to establish a point-to-point encryption.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** linked pairs of upper byte of ID and lower byte of ID

- **IS_ID_PAIRED**
  With this function it is possible to request, if the transmitted device ID is paired or not. The device ID is transmitted with the APDU parameter field of P1 and P2. This method does not have a response value; instead, the attached trailer is used to analyze the status of the request. If the trailer value correlates to the hexadecimal value of 0x9000, then the device ID is successfully paired. The device ID is not paired, when the attached trailer value represents an error code.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** no return value (only trailer)

- **GET_NEW_CMD_ACTION**
  This function requests the security controller to respond with the information, if a new control command is placed inside the buffer. If a new control command is stored in the buffer, then the stored information is included into the response. The return value is a concatenation of three byte values. The first byte represents the command type and the other two bytes defines the associated stored value.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** command type $\|$ upper byte value $\|$ lower byte value

- **AES_DEC_WITH_AES_VERIFY**
  The microcontroller receives signed and encrypted messages over the radio frequency module. In order to verify and to encrypt these messages, the security controller

provides this functionality. The microcontroller has to provide to the controller the IV for the CBC mode of the AES encryption, the encrypted message, and the MAC of the signed encrypted message. If the verification of the MAC is correct, then the return value corresponds to the decrypted message. The information for the selection of the correct cryptographic keys for the AES functions are provided by the parameters P1 and P2 by the APDU command.

Provided and returned data:

- **Input data:** AES IV ‖ Encrypted message ‖ MAC
- **Return value:** Decrypted message

- **AES_ENC_WITH_AES_SIGN**
  The microcontroller should provides the functionality to encrypt and sign new messages with the support of the security controller. This function only requests the plaintext as input to perform the process, which is transmitted with the payload of the APDU command. In addition, the security controller has to be provided with cryptographic keys that are required for the process. This information is included inside the APDU command in the parameter fields P1 and P2, and represents the device ID. The return value is a concatenation of the new created IV, encrypted message, and the MAC.

  Provided and returned data:

  - **Input data:** plaintext
  - **Return value:** AES IV ‖ Encrypted message ‖ MAC

Detailed Description of External Functions of the Applet

External functions are available at the NFC interface and enable the user to communicate with the security controller. The construction and transmission of data is protected by several points of view. One important procedure is the usage of encryption with session keys for a secured transmission of data between the security controller and the NFC-enabled device. In addition, the transmitted data is authenticated by the session keys in order to recognize if a message is modified during transmission.

- **SET_INTERNAL_ACCESS**
  With this command the microcontroller can unlock the access to only internally available functions with the knowledge of the shared secret. This command only requires the shared secret as input value for the execution. If the shared secret is valid, then the security controller enables the access to the previously explained internal functions until the Java Card applet is deselected.

  Provided and returned data:

  - **Input data:** Shared secret
  - **Return value:** no return value

- **EVAL_SESSION**
  The NFC-enabled device can check if the stored session data is still valid or if it has to be updated. This function requires an encrypted message as input. In this encrypted message byte values are randomly generated, which are encrypted with the currently available session keys. The security controller decrypts the message with its stored session keys and increments each byte in the message by one. After the incrementation process, the new message is encrypted with the same session key. The updated message is sent to the sender. The sender has to decrypt it again and check if the random generated byte values are correctly incremented by the value of one. If all bytes are correct, the NFC-enabled device and the security controller can be sure that the session keys are valid. The freshness of the session data is also protected by an incremental counter, which only allows to execute a specified number of commands. If this specified number of executed commands is reached, then this method responds to the device that the current used session is invalid.

  Provided and returned data:

  - **Input data:** Encrypted message (content of randomly generated bytes)
  - **Return value:** Encrypted message (received bytes are incremented by one)

- **CREATE_NEW_SESSION**
  This method is necessary when the previously used session data is expired or is not available any more. With this function, it is possible to generate new session keys for a secured communication between the NFC-enabled devices and the security controller. A successful execution requires the ECC public key of the requester as input. With the input of the public key, the security controller calculates new session data on the base of the defined process of ECIES. The method responds as return value the current ECC public key of the ephemeral key pair of the security controller. The public key of the controller is necessary to be able to recalculate the new session data.

  Provided and returned data:

  - **Input data:** ECC public key of requester
  - **Return value:** ECC public key of security controller

- **GET_LINKLAYER_KEY**
  In order to receive the stored keys for the link layer encryption, this function has to be executed. No input data is required for the execution of this function. In the use case of this thesis, only the gateway responds the link layer key in an encrypted message and all other IoT devices respond with an error message. The link layer key is encrypted with the current session keys, so that the key cannot be eavesdropped during the transfer.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** IV ‖ Encrypted message ‖ MAC

- **GET_NEW_PAIRING_DATA**
  This function is used to create new keys for a new point-to-point encryption. This function requires as input the new requesters device ID and optionally the link layer encryption key. The requester's device ID is transmitted with the parameter fields P1 and P2 of the APDU command. The link layer key is transmitted as payload of the APDU packet, which is marked as optional, and is only required for pairing a node with the gateway. If the link layer key is transmitted in the payload, then this payload is required to be encrypted with the session keys of the current session data. The encrypted return value of this function contains the newly created point-to-point encryption keys. In all cipher modes the present session keys for transmitted encrypted messages are always used.

  Provided and returned data:

  - **Input data:** IV ‖ Encrypted message ‖ MAC (optional payload)
  - **Return value:** IV ‖ Encrypted message ‖ MAC

- **SET_NEW_PAIRING_DATA**
  The newly created and stored point-to-point encryption keys at the NFC-enabled device have to be transmitted to the requested device in order to finalize the pairing process. This function is designed to fulfill these requirements. The function requires an input value to the new point-to-point encryption keys in encrypted way. Two contained keys are inside the payload, namely one AES key for the cipher encryption, and one AES key for the authentication. At the beginning of the message, two bytes for the device ID are additionally included, which are used for the identification of the creators ID. All this information is encrypted with the session keys to protect confidential information against attackers.

  Provided and returned data:

  - **Input data:** IV ‖ Encrypted message ‖ MAC
  - **Return value:** no return value

- **GET_ENC_DEVICE_DATA**
  The stored information about the device can be requested by this method. The stored information is encrypted and signed with the current session keys. A concatenation of a new IV for the AES cipher, encrypted content information about the device, and the signed MAC is expected as a return value.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** AES IV ‖ Encrypted message ‖ MAC

- **SET_NEW_CMD_ACTION**
  The security controller owns a dedicated buffer which is used to transmit control commands or parameters. This functionality is designed to transfer control commands from the NFC-enabled device to the security controller, which are later processed by

the microcontroller. The payload for the buffer is also encrypted by the session keys. The function call decrypts the payload data at first and is then stored inside the buffer. The encrypted message includes the construction of the control command which consists of three bytes. The first byte is defining the command type, and the other two bytes are the related command value. The microcontroller can poll the buffer to recognize if new data is inside the buffer or the external generated NFC trigger signal can be used.

Provided and returned data:

- **Input data:** AES IV ‖ Encrypted message ‖ MAC
- **Return value:** no return value

- **GET_NODE_TYPE**
  This function is the equivalent for setting the node type. The return value is the current node type with the same device encoding as the "SET_NODE_TYPE" function.

  Provided and returned data:

  - **Input data:** no additional information
  - **Return value:** node type value (0 = default, 1 = gateway, 2 = node)

Enhanced Error Codes

In the implementation of the above discussed functions, new error codes are added for correct error handling. The new added error codes are listed and shortly described by their meaning in Table 4.3. These error codes are constructed to be placed in the proprietary area so they can be compatible with the other APDU errors.

Secured Storage Management

Another use case of the security controller is to store IoT device specific information in a secured NVM. The advantage of this storage place is that the memory is protected against side channel attacks and other cryptographic attacks. One part of the entire available memory is reserved to store cryptographic keys of the paired devices, and the second part is partitioned for the IoT device specific content data.

The device specific content can only be received from the security controller when a valid session is created between the security controller and the participant. A memory storage of 200 bytes is reserved for the content data. If this amount of data is not enough, it is possible to extend it in the security controller with a simple definition.

The storage place for the paired cryptographic keys is organized in an array structure. The positions of the array are mapped pairwise to a paired device ID. For this organization, two arrays are used. One array holds all cryptographic keys, and the second one maps the keys to the paired device ID. The point-to-point cryptographic keys for the gateway are stored in the IoT devices always on the position 0 and 1. The first element of a key pair is the AES cipher key, and the second position represents the AES key for the

Name	Error code	Description
AES_ERROR	0x6510	In the cipher execution occurred an error.
AES_SIGN_ERROR	0x6511	In the sign process occurred an error.
CMD_NOT_FINISHED	0x6512	The control command has not picked up.
CMD_FINISHED	0x6513	No control command is stored in the buffer.
NODE_ID_NOT_FOUND	0x6514	Device ID has not been found in the list of paired devices.
PAIRING_NO_FREE_SLOT	0x6515	The buffer for storing the information of paired devices is full.
PAIRING_SAME_ID	0x6516	Execution cannot be forced on the same device.
ACCESS_DENIED	0x6517	Access is denied to internally functions.

Table 4.3: Additionally defined error codes beside the standard APDU ones.

authentication process. All other IoT device keys are stored after the gateway in the same order. Figure 4.15 depicts the organizational memory structure of the stored keys inside the array.

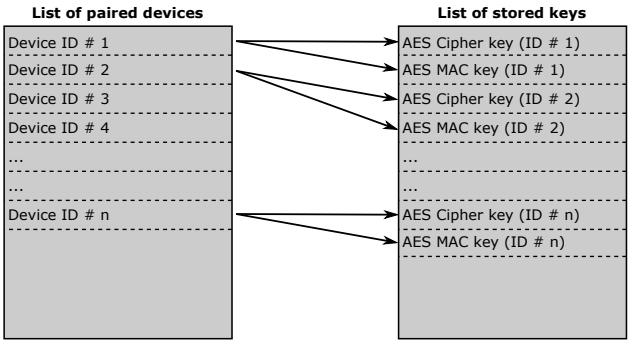


Figure 4.15: Configuration of storage mapping for paired key information inside the security controller.

4.3.4 Designed Android Application

The Android application with the name “Secure Smart Home” is the main part of the defined secured pairing process between two communication participants. This application has to meet the design requirements with the defined activities. The conclusion out of the design process showed, it is necessary to implement four miscellaneous activities for this application. In the implementation phase, state diagrams are created for each activ-

ity, which are subsequently implemented into the Android application. The next section describes each activity in detail based on their functionality and composition.

Main Activity

In the main activity, the user is able to scan any deployed IoT device for receiving all essential information by the support of NFC. For receiving this information in a secured environment, the Android device, which must be equipped with NFC, has to follow the defined working process of the security controller to establish a new session. Figure 4.16 illustrates the state diagram of the main activity. In the beginning, the application has to check if the session data is available. If the session keys are not present or not valid, then the Android device has to force the process to create new session keys, which is based on the concept of ECIES (described in Section 3.5.1). If all session keys exist and are valid now, then the NFC-enabled device can trigger an event for receiving encrypted content data from the scanned IoT device. The received data from the security controller is encrypted with the current valid session data.

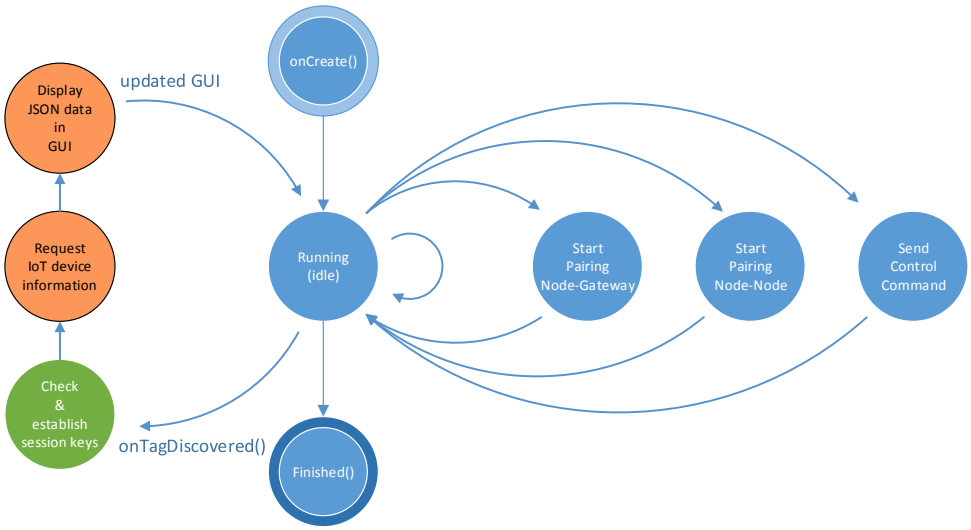


Figure 4.16: State diagram of main activity in Android.

Furthermore, the main activity provides several functionalities to the user, which are visible after a successful scanning process of an IoT device. These functionalities for the specified IoT device can be to send control commands to the security controller, force a new pairing process between the gateway or another node, and to perform configurations. The set of functionalities depends on the scanned node type.

Figure 4.16 depicts a green state with the label “Check & establish session keys”. This state contains several sub states, which check and verify if the current stored session keys are still valid. Figure 4.17 illustrates the previously described process of checking and establishing new session keys.

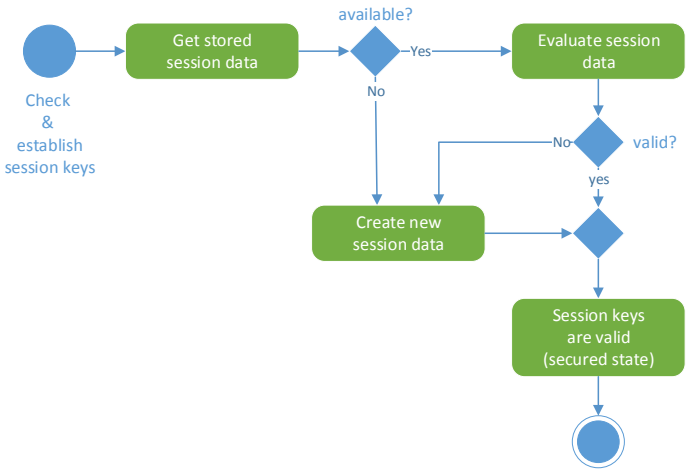


Figure 4.17: State diagram of verifying session keys in the Android application.

Send Control Command to IoT Device

This activity is designed for transmitting control commands to the IoT devices. This process requires to follow a specific flow of actions to send the commands in a secured way to the device. This activity is useful to force actions or to perform configurations on a specified IoT device. Figure 4.18 depicts the state diagram of the described activity. In the first step, it is required to check and validate the session data following the same workflow as it is depicted in Figure 4.17. If the session data is correct, then the control command is built up with the selected command type and value. The constructed message is subsequently encrypted with the present session keys, and is transmitted to the IoT device via the NFC communication channel.

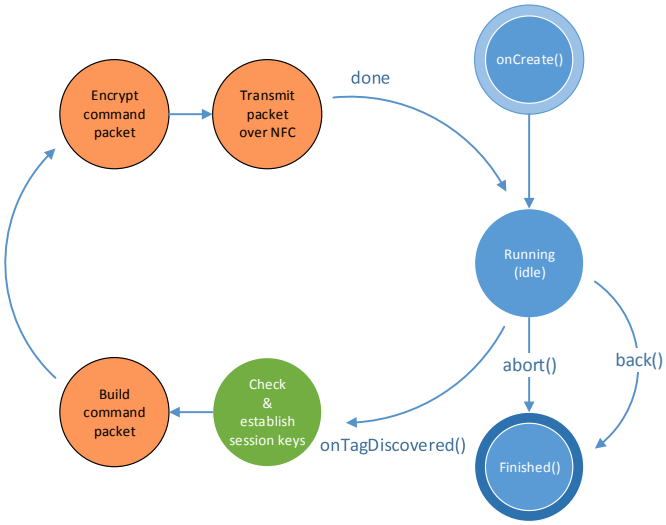


Figure 4.18: State diagram of processing and transmitting a control command to an IoT device.

Pairing Process Between Two IoT Devices

The pairing process between two IoT devices is necessary in the designed security concept in order to exchange the cryptographic keys in a secured way. Two versions of the pairing process are available, and the version selected depends on the participating device. One workflow exists for the combination gateway and node, and an additional one exists for two nodes. The main differences between these two workflows are in the amount and types of exchanged cryptographic keys. The detailed processes of the various pairing scenarios are described below in the following section.

- **Pairing Between Node and Gateway**
  Figure 4.19 depicts the state diagram of the pairing process between a gateway and a node. The first stage is to check if session data is available and valid, followed by the process defined in Figure 4.17. If this process is followed, then the user has to scan the gateway for receiving the link layer encryption key. This action is summarized in part 1 of the state diagram in Figure 4.19. In part 2, the node, which started the process, has to be scanned again with the NFC-enabled device in order to transmit the received link layer key, and for requesting to create new key pairs for the point-to-point encryption. The last step in the pairing process, part 3, is to transmit the new created key pairs to the gateway with a rescan of the gateway via NFC. After all these steps, the gateway is successfully paired with the node and the communication is encrypted and authenticated.

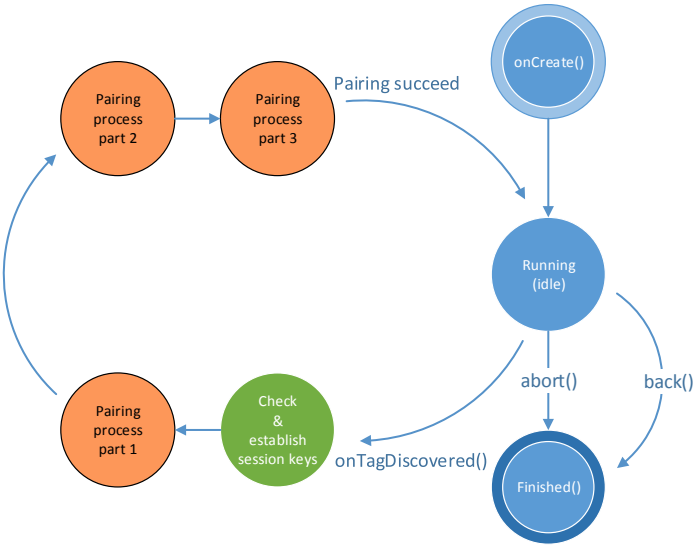


Figure 4.19: State diagram of pairing process between a node and a gateway.

- **Pairing Between Two Nodes**
  The process of pairing two nodes is nearly the same than the process for the gateway. Figure 4.20 illustrates the detailed state diagram of the pairing process. The state machine in the Android application has to manage the different states for a successful pairing. In the first stage, new point-to-point encryption key pairs by the second node are created, which want to be paired with the original node. The newly created cryptographic keys are transmitted to the Android device, which is represented in part 1. After the keys are received, the user has to scan the original node to transmit the new key pairs of the new paired IoT device. The last described step is labeled in the state diagram as part 2. After these two steps, the two nodes can communicate in an isolated channel with each other. The channel is encrypted and authenticated.

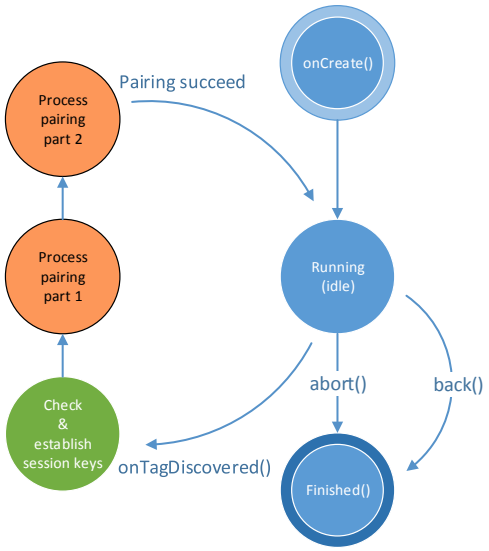


Figure 4.20: Pairing process between two IoT nodes. State diagram demonstrates the required steps inside the Android application.

Chapter 5

Results

In this chapter, the complete designed IoT environment for a secured smart home application is presented. The first part of this chapter focuses on the construction of the hardware devices and their interaction in the entire system environment. The security concept of the network is recapped and all implemented security mechanisms are pointed out. Lastly, the end-user platforms are described by their appearance and their usability.

5.1 Evaluation of Interacting Components

The entire IoT environment consists of several devices and components that construct a secured smart home application. The two main devices inside the IoT construction are the gateway and the distributed IoT nodes. In the demonstration setup of the secured smart home application, one gateway, one smart switch, and several smart outlets are used. The gateway, as border router, is connected to a standard home router to provide the accessibility to the WWW. The home router is equipped with WiFi to enable to connect the tablet and smartphone to the network.

5.1.1 Assembled Smart Home Demonstrator

Figure 5.1 depicts the constructed secured smart home application. On the left side of the Figure, a smartphone and a tablet are connected to the gateway via the WiFi communication channel of the home router. Those devices have the possibility to access the webserver of the gateway and to display the hosted website with the network address "http://192.168.1.1/index.htm". In the middle of the Figure is the gateway with the special assignment to translate the different physical communication environments to the other ones, for example; from the WSN to the WWW network and vice versa. The IoT nodes with typical functionalities for a smart home environment are on the right side. Furthermore, this section focuses on each constructed IoT device, depicting their resulted functionality and construction sizes.

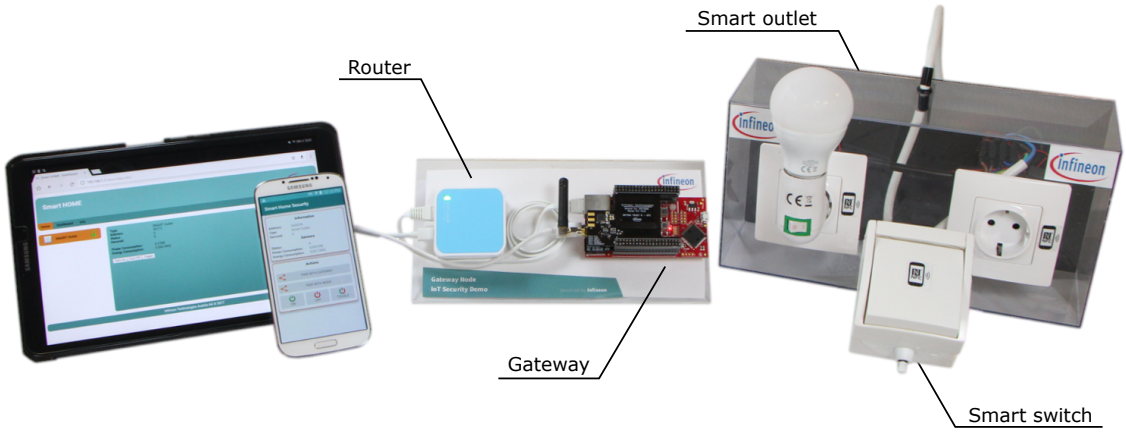


Figure 5.1: Entire secured smart home application setup of the designed application.

5.1.2 Gateway with Security-Enhancements

Figure 5.2 illustrates the constructed gateway with all requested external hardware components. At the base of the gateway is the board with the main components, the evaluation kit XMC4500 Relax Kit, which provides the device with the base functionality. The base functionality of this evaluation kit is the support of Ethernet, SD Card, microcontroller, and debugging system. On the evaluation kit, two add-on shields are stacked to extend the basic functionality. The first add-on shield, counting from the bottom to the top, provides the base system with a transceiver module for the WSN environment. The second add-on shield on the top enhances the complete gateway with the support of a security controller. A NFC antenna is located on this add-on shield with the security controller, enabling a direct communication interface to NFC-enabled devices.

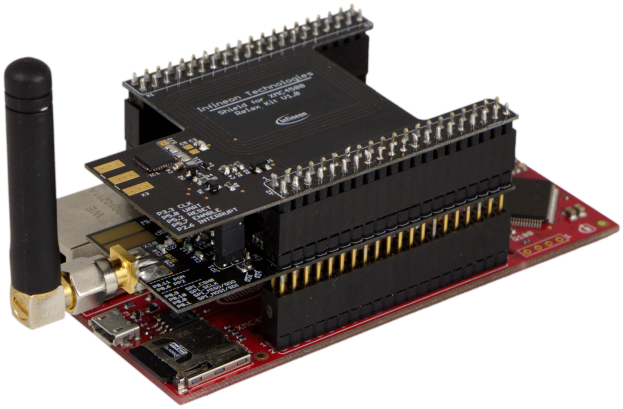


Figure 5.2: Constructed gateway for the usage in the IoT environment of a secured smart home application.

Challenges of the Add-on Shield - Security Controller

In the implementation phase of the add-on shield with the security controller, an issue is detected with the power management unit in a common operation mode. The power management unit is designed to switch the power supply of the security controller with a high side MOSFET switching circuit. The issue with the designed circuit occurs in normal operation when the microcontroller tries to switch off the power supply to the security controller. The misbehavior appears in a way that power supply is not disconnected when the enable pin of the power management unit is set to low. In a closer inspection of the voltage values of the circuit, it appears that the output voltage of the power management unit is approximately 2.7V instead of 0V. Consequently, the misbehavior is investigated in more detail in the current operating mode to find the error.

At first, the current operation mode is analyzed and focuses on the circuit construction and their input values. In the present mode, the values of the input pins on the security controller are observed. The reset line (RST) is powered with 3.3V and the other inputs are pushed to ground. The reset line is powered with 3.3V, because this input line is Active-Low and is consequently initialized to be always in high state in idle mode. In the first observation of the constructed circuit, no design mistakes are detected. Due to this reason, the analysis has to be processed in more detail. Therefore, the answer is searched in the input structure of the security controller due to the reset line with the powered input. Figure 5.3 depicts the evaluated input circuit with the power management unit. The reason for the connected MOSFET is that the MOSFET is not able to operate in a defined cut off region. In addition, the Figure illustrates the use case of the propagated voltage through the reset line over the Electrostatic Discharge (ESD) protection circuit to the power supply (VDD) pin of the security controller. Consequently, the power supply pin is driven by some volts through the reset line and the ESD protection circuit. The consequence of this operation is that the MOSFET works in a linear mode, and the security controller cannot be totally disconnected from the power supply.

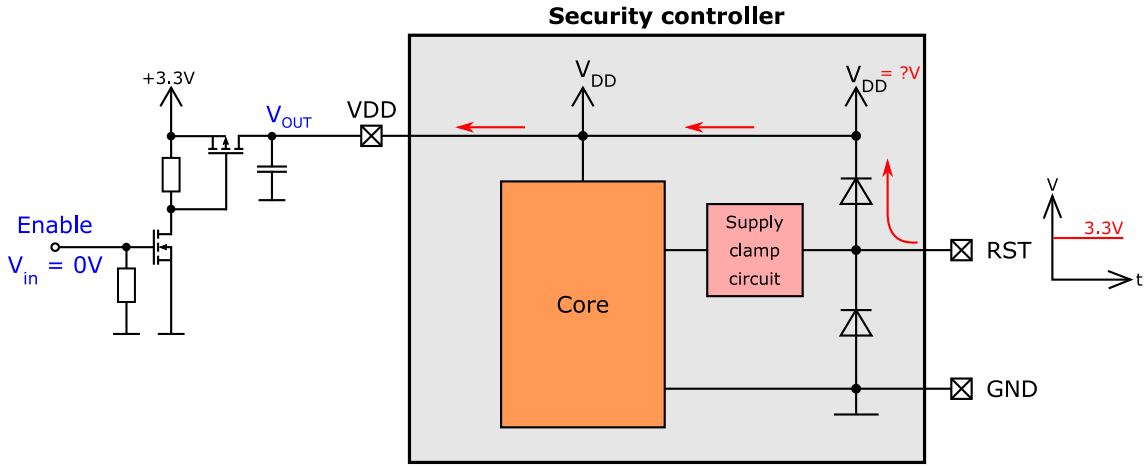
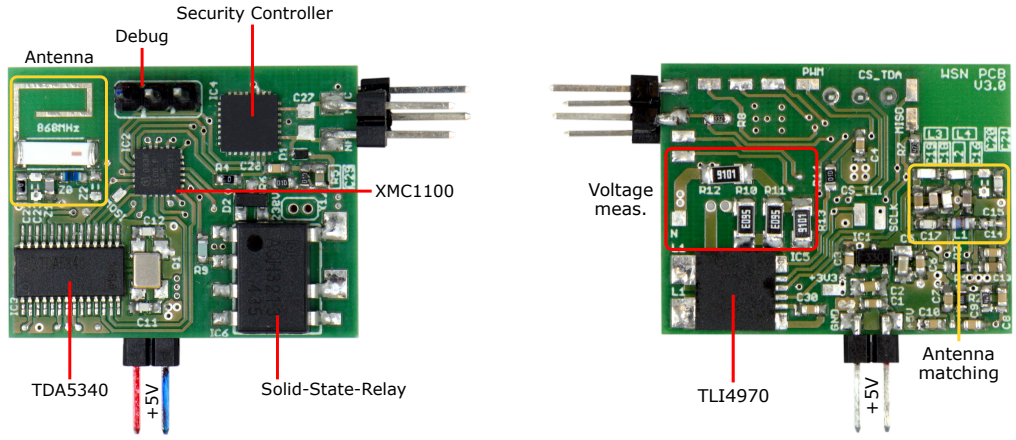


Figure 5.3: Illustrated misbehavior of the power management unit for the security controller.

The solution for this issue is to switch all inputs of the security controller to ground before the power management unit is able to switch off the power supply of the security controller. In the power down mode, the input pins are not allowed to power with a defined high state because this action would lead to an undefined operating mode of the security controller, and the NFC communication interface would be disabled all the time.

5.1.3 Redesigned IoT nodes

The designed hardware platform for the IoT nodes is depicted in Figure 5.4. In the implementation phase, it is pointed out that the constructed PCB should either be usable for the smart outlet or for the smart switch platform. Figure 5.4 shows the PCB board of the IoT node from the top and bottom view. The total dimensions of the PCB are in lengthwise direction 31mm and in width 24mm. On this PCB, all required hardware components for both platform versions are placed. The placed components enable the device to communicate with the WSN with the on-board placed antenna. Furthermore, the IoT node is equipped with several sensors and actuators to perform the defined tasks. Some of the sensors and actuators are the current sensor (TLI4970), relay, and voltage measurement circuit. In Figure 5.4 is supplementary labeled the various hardware components.



(a) Top view **(b)** Bottom view

Figure 5.4: These two Figures show the constructed IoT node from the top and bottom view. This IoT node is designed to be able to operate as smart outlet or as smart switch.

5.2 Evaluation of New Security Concept

The security concept of the smart home application is constructed with several cryptographic techniques. These techniques are distributed at miscellaneous entry points of the entire IoT environment. Figure 5.5 demonstrates all implemented security features inside the constructed architecture in order to gain a satisfied security level in relation to energy consumption.

On the first view it is apparent that only the communication inside the WSN environment is protected with several cryptographic mechanisms. The other part, exactly from the gateway to the WWW, goes beyond the responsibility of this master thesis. The focus is lead to the WSN part of the IoT environment. The WWW environment can be enhanced with security through extending the webserver with a TLS encryption.

In the WSN environment, the following security enhancements are introduced. The numbering corresponds to Figure 5.5:

1. **Link Layer Encryption**
   The entire WSN is protected by a link layer encryption, which is a part of the 6LoWPAN protocol stack inside the Contiki OS environment. The link layer protection uses AES cipher as cryptographic algorithm.

2. **Communication Between IoT Devices**
   The communication between the IoT devices is encrypted with a point-to-point encryption to ensure a secured transfer of data. A point-to-point encryption is established between all nodes to the gateway and also between some nodes. The channel is encrypted and authenticated; therefore, for this procedure, an AES cipher in CBC mode is used. The key length for the cipher is defined to 128 bits.

3. **IoT Node**
   The developed IoT nodes are equipped with a hardware secured element. This hardware secured element provides the platform with a hardware acceleration for cryptographic algorithms, and a protected NVM storage. Information and configurations are stored on this memory storage of the IoT nodes. The secured element is constructed to be resistant against side-channel attacks and consequently provides a high security level for the nodes themselves.

4. **NFC Communication**
   NFC technology is used for the first key exchange to force the pairing tasks in a secured way. The workflow is implemented in an Android application, which transfers the data to the IoT devices with the support of ECC. The ECC in ECIES mode is used, and requires a public/private key pair with a key length of 256 bits.

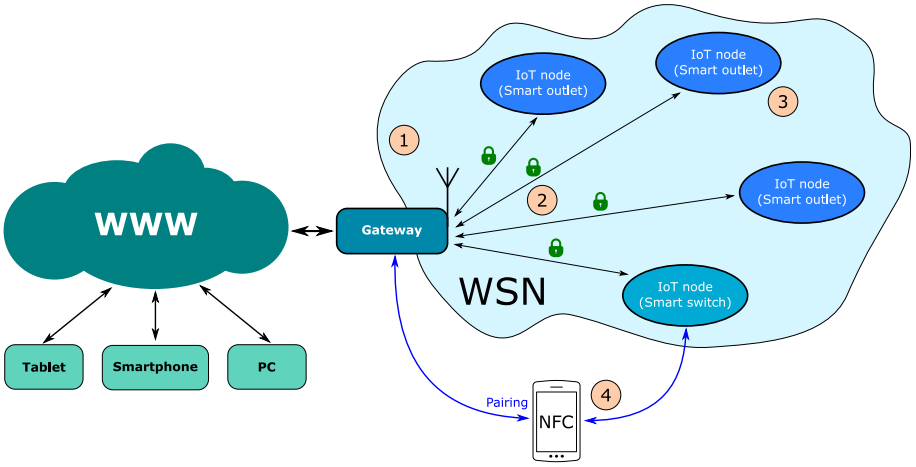


Figure 5.5: Illustrates the developed security concept for the secured smart home application. Additional labels include the enhanced security features of the IoT environment.

5.2.1 Payload Overhead due to Security Enhancements

IoT nodes communicate over the WSN with the gateway to exchange information or other required data. The communication channel is secured with an AES cipher in CBC mode. Consequently, every data packet is separately encrypted, and produces additional overhead in comparison to an unencrypted transmission. Table 5.1 demonstrates the structure of one encrypted payload packet. The interpretation of this Table leads to a minimum payload length for one encrypted message to 384 bits with an informative content transfer of a maximum of 128 bits. Consequently, the resulting overhead is 254 bits, which leads to an overhead consumption of 200% in comparison to a plaintext transmission.

Type	AES IV	Encrypted Payload	MAC
# of bits	128	n * 128	128

Table 5.1: Resulting payload length for one encrypted message block. In the encrypted payload field “n” is the number of blocks of sizes of 128 bits.

In the design chapter in section 3.5.1, the process of using ECC cryptography for the encryption is discussed. In the evaluated use case of using ECC in the encryption process based on ECIES, this demonstrates that the generated overhead is increased as well in comparison to the AES cipher.

Nevertheless, the overhead qualifies the process of transmitting the payload to the other participants in an encrypted and authenticated way. The conclusion of the security enhanced implementation is that each activated security feature conducts more overhead, but leads to a higher security level and protects the privacy of the users.

5.3 Android Application Evaluation - Smart Home Security

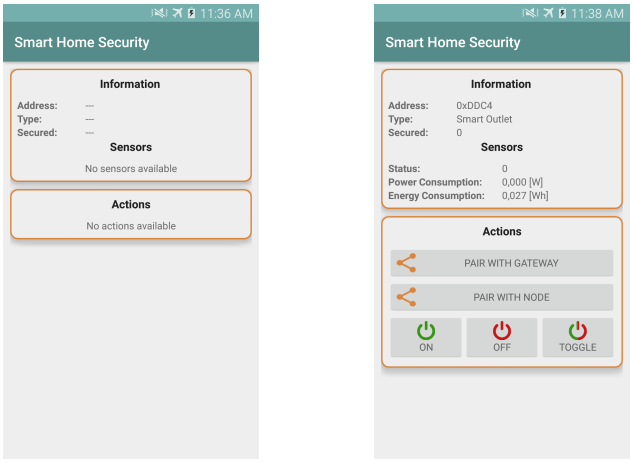
The application for Android devices is designed for several use cases. These use cases are for visualizing IoT device information, to send control commands, and to proceed the pairing between other devices. One of the main activities is the process of pairings. The graphic design of the GUI is kept simple and user friendly. The usage of the application should be self-explanatory that guide the user through the different activities.

5.3.1 GUI Design

This section focuses on the designed GUI for the secured smart home IoT environment. The documentation describes the provided functionality of the various activities and the visualization.

Main Activity

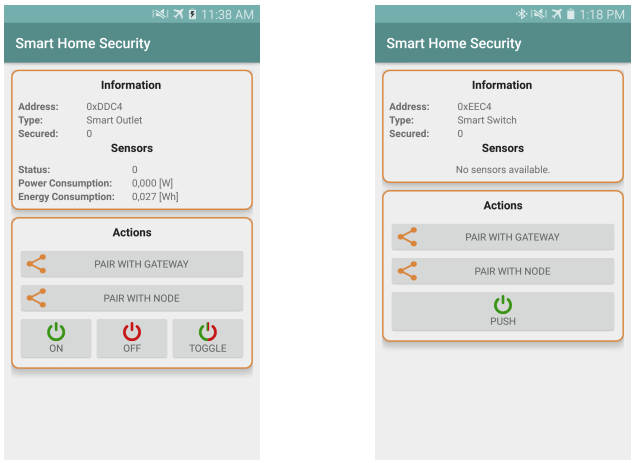
The main activity of the application provides the user with the possibility to scan each IoT device inside the environment with NFC technology and to receive the stored device information. Figure 5.6 depicts the different appearances of the main activity screen of the application. The left Figure 5.6a demonstrates the appearance of the application directly after the start. If a device is scanned with the support of NFC, then the application changes the appearance by displaying the received device content. The described illustration is depicted in Figure 5.6b. This Figure shows the separation of the application into two main groups. The upper part of the screen displays the general information of the scanned IoT device. The general information is followed by the available sensors including their current measured values. In the last group, IoT device specific commands are located, controlling the device via NFC with the functionalities of sending control commands, and to force pairings between devices.



(a) Start screen **(b)** Scanned IoT device

Figure 5.6: Android application in main activity. Figure (a) depicts the appearance after starting the application. Figure (b) shows the result of a scanned IoT device.

Figure 5.7 depicts two different visualizations from the main activity when different IoT devices are scanned. The difference between these two illustrations is in the amount of attached sensors, and on the available set of action commands. Figure 5.7a shows the use case when a smart outlet is scanned. This Figure illustrates the measured sensor values of power and energy consumption. The set of action commands contain methods to force the process of pairing and to change the output state (on, off, toggle) of the outlet. Figure 5.7b demonstrates the use case of a smart switch. The smart switch contains less sensor values than a smart outlet, and only provides an action command to virtually push the button over the NFC communication channel.



(a) Smart outlet **(b)** Smart switch

Figure 5.7: Illustrates the two different appearances of the application, when different IoT devices are scanned. The appearance of the application changes only in the amount of sensor values and in the set of action commands.

Pairing Process

The process of pairing is separated into two different workflows. One workflow is for the process of pairing one node with a gateway, and the other one for pairing two nodes. The process of pairing two nodes allows the user to link two different IoT nodes with diverse functionalities together. The result of this pairing process is that one node can control the other node. For example, if a smart switch is paired with a smart outlet, then the smart switch has the permission to send control commands over the WSN to the smart outlet in order to change the output state. In addition, this pairing is used to secure the connection between the two devices with an additional point-to-point encryption. Figure 5.8 demonstrates the various visualizations of the workflows. In the middle of the entire illustration, Figure 5.8b depicts and highlights the two essential buttons which starts the process of pairing. Figure 5.8a illustrates the screenshot of the workflow for the pairing between a gateway and a node. Figure 5.8c illustrates the workflow for the pairing between two IoT nodes.

(a) Gateway/IoT node **(b)** Start screen **(c)** Two IoT nodes

Figure 5.8: Demonstrating the appearances of the Android application with the implemented pairing processes.

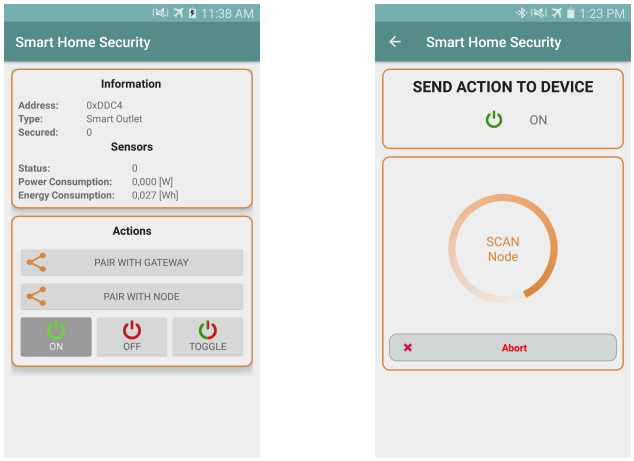
NFC-based IoT Device Interaction

Figure 5.9 illustrates the process of sending a control command over NFC to a desired IoT node. For this operation the desired IoT node has to be scanned in order to result in what is illustrated in Figure 5.9a. Now, the user can choose from a list of action commands. The set of commands depends on the type of the IoT node. Figure 5.9a illustrates the information of a smart outlet with the basic functionality of switching the output of the device on, off, or to toggle it. Figure 5.9b illustrates the appearance of the view of the send control command activity of the application. The user is now able to scan the chosen IoT node, which forces the execution of the chosen command. The Android application automatically returns to the main view when the command is successfully transmitted to the node.

5.3.2 Usability Analysis

The usability analysis focuses on the process of pairing new devices. As mentioned before, two different pairing processes exist due to the different devices. The process of enabling security in an IoT environment must be as simple as possible and should not provide space for misuses. Most people do not have detailed information about security, and also not about functionality. Consequently, the importance of activating the correct usage is commonly underestimated. Nevertheless, the sense for the security topic slowly increases at the user side and they understand the importance of usage of security features. If the process of activating the security enhancements is too complex for the end user, then the process is aborted after some steps or is completely skipped from the beginning.

These facts lead to the use of a smartphone with NFC technology to exchange the cryptographic keys in an easy and protected way. The user is guided through all steps with short descriptions of the tasks. The cryptographic keys are exchanged with the security



(a) Main activity **(b)** Send action to node

Figure 5.9: Demonstration of the use case to transmit a control command to the desired IoT node. Figure (a) depicts the main activity with the visualization of a scanned smart outlet device. Figure (b) illustrates the screen of the device, when an "ON" command is selected to transmit to the node.

concept of ECC methodology between the smartphone and the security controller. These two workflows of the pairing process only vary in the amount of exchanged keys. The pairing between a node and a gateway requires three steps of user interaction. In the first step, the present link layer encryption key of the gateway is received. This link layer key is checked each pairing on the gateway by the smartphone because the time between pairing several devices, it may happen that the link layer encryption key is already updated. Consequently, the new node would not be able to communicate inside the IoT environment. In the second step in the pairing process, the link layer key to the desired IoT node is transmitted, and the node subsequently creates new cryptographic keys for the point-to-point encryption. In the last step, the smartphone transmits the processed point-to-point keys to the gateway. All communication activities are performed over a NFC communication channel.

The pairing process between two nodes requires only two steps because the exchange process of the link layer key is excluded. The exclusion of this key is possible because the IoT nodes are already paired inside the IoT environment with the gateway.

In short, the processes for activating the security enhancements only require the user to perform a maximum of three operations. These operations are very simple by only scanning the desired IoT device with the NFC-enabled device. The workflow, which is illustrated in the Android application, guides the user through all steps and informs the user for incorrectly performed operations.

5.4 Evaluation of Website for Dynamic Device Management

The website has the responsibility to provide up to date information of the current IoT environment. Figure 5.10 illustrates the implemented website with a detailed view of the "home" site. A list of all available IoT nodes is located on this site, which are in the communication range of the gateway. The list of nodes provides additional information of the connected nodes like if they are paired with the gateway or not. The main content area of the website visualizes the received IoT node information. In this area of the website, it is also possible to send control commands to the selected node.

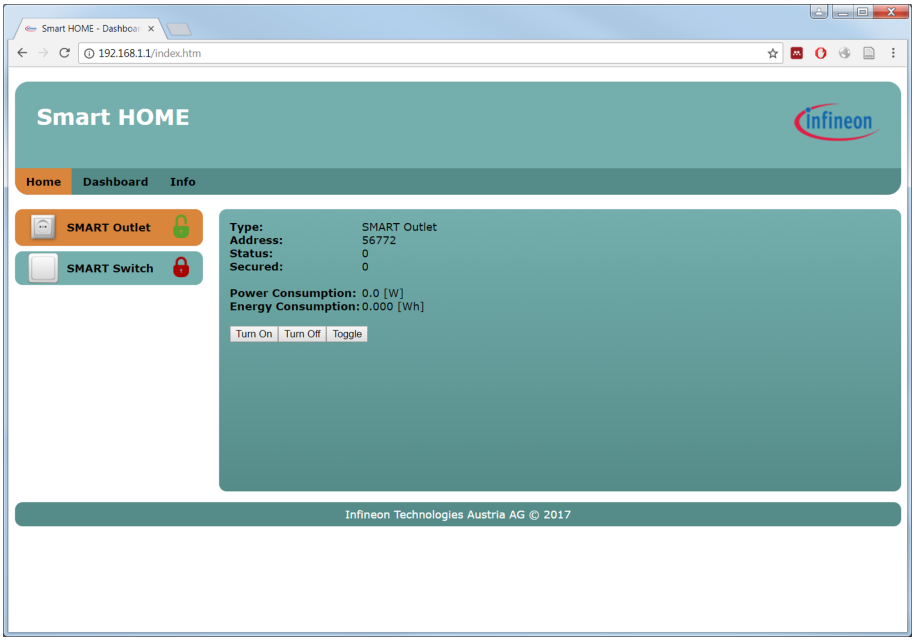


Figure 5.10: Website in "home" view with two connected IoT nodes, one smart switch, and one smart outlet. Left side shows the list of currently connected IoT nodes, and on the right side the detailed information of the selected node is illustrated.

This website is able to dynamically interact with added or removed IoT devices. Furthermore, this website handles and shows encrypted communication between the IoT devices. In principle, this website is still backward compatible with earlier Contiki OS-based demonstrators without the security features, which are developed in this master thesis.

Chapter 6

Conclusion and Future Work

This chapter summarizes this entire master thesis with the achieved results of security enhancements of a smart home application. The outlook and potential future work of the thesis is also discussed, based on the developed and implemented secured smart home application.

6.1 Conclusion

The literature review of IoT, including WSN, showed that these subjects have already a better understanding for security and also provides security features inside the used network protocol standards. Nevertheless, the initial key exchange is an important point in the establishment of a secured IoT environment. Presently available standards use various techniques to transfer them in a secured way, including QR codes and button presses to identify the devices.

In this master thesis, the security enhancements at several points in the IoT environment are implemented. An essential point of these enhancements is the introduction of a point-to-point encryption between the involved network participants inside the WSN. The important point of transmitting processed cryptographic keys between the participants is done with the support of NFC technology. The standard configuration of NFC does not provide a secured communication. In order to overcome this problem, the communication is separately secured by ECC with the concept of ECIES. An advantage of this procedure is that an encrypted message is directly transmitted over NFC to the Android application and the application itself is responsible for decrypting the message. Consequently, the communication cannot be interfered by someone, due to the protection in term of encryption. QR codes, in comparison to NFC, contain a fixed information which is transmitted over the camera module to the application. On this way to the application it is possible to manipulate the QR code, due to a plaintext transmission. An additional advantage of the usage of NFC is that the IoT device can also be configured and integrated into the IoT infrastructure before the device is powered for the first time. This opportunity is only possible through the NFC feature to power devices inside the electromagnetic field. Consequently, the devices can operate right away inside a secured environment.

In the implementation phase, the usage of Contiki OS as an embedded OS showed that the usage of the included features are not running out of the box, and require a lot of

work to be runnable for the requested needs. The development of the end-user software components, such as the website and the Android application, provide the user an easy entry to the world of IoT infrastructure and to use the provided features in an easy way. The user has the option to use either the computer, tablet, or smartphone to control and manage the IoT network with the implemented website or directly with a NFC-enabled device. The website is directly hosted by the gateway and do not require a connection to a cloud service. The advantage of this procedure is that all sensitive data is not transmitted into the cloud, where nobody knows exactly where they are stored. In this smart home application, all data is stored directly in the security controller on the gateway side. Consequently, the privacy of the customer is secured from several perspectives.

In the design process, different methods of securing the communication between two IoT devices are evaluated. The evaluation was focused on the total resulting length of a typical payload packet. The resulting length of one payload packet is directly related to energy consumption because of the longer time for transmission and cryptographic methods. For the point-to-point encryption, an AES cipher in combination with a CBC mode is used. The used key length is 128 bits. The communication between the participants is not only encrypted but also authenticated by an encrypted MAC.

6.2 Future Work

With the presently developed secured smart home application, including the integration of a security controller, an application is demonstrated which uses enhanced security features. The entire IoT environment uses cryptographic methods to provide an adequate security level. On the basis of the developed and constructed secured smart home application, it is now possible to evaluate various approaches for a secured environment. The next section summarizes future work in relation to general topics for the IoT environment, including enhancing security. Interesting features for a smart home application are:

- **Energy Efficiency**
  A necessary point in the IoT nodes would be to focus in detail on the energy efficiency of the overall device level, especially in the implementation inside the embedded OS. The OS provides functionality to directly save energy by deactivating various hardware components during the time when they are not required for a normal operation.

- **NFC Technology Inside IoT**
  The present implementation uses a smartphone to exchange the cryptographic keys and consequently handles the configuration of the devices. This procedure can further be simplified by removing the smartphone as a transmission resource and uses the gateway as communication participant. On the gateway, a NFC technology can be implemented in reader mode in order to directly communicate with the IoT node, which are constructed as simple tag devices. With this enhancement, a new IoT device only has to be held inside the communication range of the NFC field of the gateway, and the gateway would complete the workflow for integrating and configuring a new device.

- **PKI (Public Key Infrastructure) Methods**
  In the presented approach, a PKI was not applicable for the communication inside the WSN in a smart home application, due to the resulting long payload sizes for only small data packets. Furthermore, the higher assumed energy consumption should be verified with a direct comparison of different algorithms. The resulting overhead by the usage of a PKI algorithm is at this point in time too high for the communication inside a WSN. This subject was discussed in detail for ECC in Section 3.5.1.

- **Additional Use Cases for the Security Controller**
  In the implementation process, additional use cases have been discovered for the security controller. These use cases are to provide a secured environment for transferring secured firmware updates to the microcontrollers. With the support of the security controller the firmware can be encrypted and verified in hardware. In addition, the firmware could be transferred using the WSN communication channel, or with a NFC-enabled device. Having a defined process for updating firmware in a secured way is important from the point of view of the customer to be able to update all system components with the latest firmware to patch security issues in the IoT devices.

Appendix A

Acronyms

6LoWPAN	IPv6 over Low Power Wireless Personal Area Network
AC	Alternating Current
ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript and XML
AM	Amplitude Modulation
API	Application Program Interface
APDU	Application Protocol Data Unit
BLE	Bluetooth Low Energy
CBC	Cipher Block Chaining
CL-EKM	Certificateless-Effective Key Management
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
COM	Component Object Model
DC	Direct Current
DoS	Denial of Service
DSK	Device-Specific Key
EBS	Exclusive Basis System
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman

ECIES Elliptic Curve Integrated Encryption Scheme

EDDK Energy-efficient Distributed Deterministic Keymanagement

EEKM Energy-Efficient Key Management

ERU Event Request Unit

ESD Electrostatic Discharge

FIFO First In First Out

GUI Graphical User Interface

HMAC Hash-based Message Authentication Code

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

I^2C Inter-Integrated Circuit

ICMP Internet Control Message Protocol

ID IDentification

IDE Integrated Development Environment

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

ISM Industrial, Scientific and Medical

IV Initialization Vector

JSON JavaScript Object Notation

KA Key Agreement

KDF Key Derivation Function

LASSB Location-Aware and Secret Share Based

LEAP Localized Encryption and Authentication Protocol

LED Light Emitting Diode

LOCK Localized Combinatorial Keying

MAC	Media Access Control
MAC	Message Authentication Code
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
NFC	Near Field Communication
NVM	Non-Volatile Memory
OS	Operating System
OSI-Model	Open Systems Interconnection Model
OTMK	Opaque Transitory Master Key
PCB	Printed Circuit Board
PKI	Public Key Infrastructure
PWM	Pulse-Width Modulation
QR	Quick Response
RAM	Random-Access Memory
RF	Radio Frequency
RFID	Radio-Frequency IDentification
ROM	Read-Only Memory
RPL	Routing Protocol for Low-Power and Lossy Networks
RMS	Root Mean Square
RSA	Rivest-Shamir-Adleman
SD Card	Secure Digital Card
SDR	Software-Defined Radio
SHELL	Scalable, Hierarchical, Efficient, Location-aware and Lightweight
SIM	Subscriber Identity Module
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
STKM	Spanning Tree Key Management
TCP	Transmission Control Protocol
TLS	Transport Layer Security

TRNG True Random Number Generator

UART Universal Asynchronous Receiver Transmitter

UDP User Datagram Protocol

USB Universal Serial Bus

USIC Universal Serial Interface Channel

WiFi Wireless Fidelity

WSN Wireless Sensor Network

WWW World Wide Web

Bibliography

[1] *Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says — IHS Online Newsroom.* URL: `http://news.ihsmarkit.com/press-release/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says` (visited on May 26, 2017).

[2] S. S. Solapure and H. Kenchannavar. „Internet of Things: A Survey Related to Various Recent Architectures and Platforms Available". In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Sept. 2016, pp. 2296–2301.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. „The Internet of Things: A Survey". In: *Computer Networks* 54.15 (Oct. 28, 2010), pp. 2787–2805. ISSN: 1389-1286.

[4] Divya Sharma, Sandeep Verma, and Kanika Sharma. „Network Topologies in Wireless Sensor Networks: A Review". In: *International Journal of Electronics & Communication Technology (IJECT)* 4.3 (2013). ISSN: 2230-9543.

[5] Michal Kerndl. „Energy Harvesting Using Nanoegenerators in the Internet of Things". URL: `http://www.academia.edu/24956062` (visited on Oct. 29, 2017).

[6] S. Ahmad Salehi, M. A. Razzaque, P. Naraei, et al. „Security in Wireless Sensor Networks: Issues and Challanges". In: *2013 IEEE International Conference on Space Science and Communication (IconSpace)*. 2013 IEEE International Conference on Space Science and Communication (IconSpace). July 2013, pp. 356–360.

[7] Dr G. Padmavathi and Mrs D. Shanmugapriya. „A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks". In: *International Journal of Computer Science and Information Security (IJCSIS)* 4.1 (Aug. 2009). ISSN: 1947 5500.

[8] S. Shanthi and E. G. Rajan. „Comprehensive Analysis of Security Attacks and Intrusion Detection System in Wireless Sensor Networks". In: *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*. 2016 2nd International Conference on Next Generation Computing Technologies (NGCT). Oct. 2016, pp. 426–431.

[9] D. R. Raymond and S. F. Midkiff. „Denial-of-Service in Wireless Sensor Networks: Attacks and Defenses". In: *IEEE Pervasive Computing* 7.1 (Jan. 2008), pp. 74–81. ISSN: 1536-1268.

[10] M. Brownfield, Yatharth Gupta, and N. Davis. „Wireless Sensor Network Denial of Sleep Attack“. In: *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. June 2005, pp. 356–364.

[11] C. Karlof and D. Wagner. „Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures“. In: *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.* Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003. May 2003, pp. 113–127.

[12] J. Newsome, E. Shi, D. Song, et al. „The Sybil Attack in Sensor Networks: Analysis Defenses“. In: *Third International Symposium on Information Processing in Sensor Networks, 2004. IPSN 2004.* Third International Symposium on Information Processing in Sensor Networks, 2004. IPSN 2004. Apr. 2004, pp. 259–268.

[13] *Zigbee Alliance*. URL: `http://www.zigbee.org/` (visited on Oct. 28, 2017).

[14] ZigBee Alliance. *ZigBee Specification 2005*. URL: `https://www3.nd.edu/~mhaenggi/ee67011/zigbee.pdf` (visited on June 5, 2017).

[15] ZigBee Alliance. *ZigBee Specification 2006*.

[16] ZigBee Alliance. *ZigBee Specification 2007*.

[17] Oliver Hersent, David Boswarthick, and Omar Elloumi. *The Internet of Things: Key Applications and Protocol*. 2. Wiley Publishing, 2012. ISBN: 1-119-99435-7. URL: `http://www.wiley.com/WileyCDA/WileyTitle/productCd-1119994357.html` (visited on Oct. 29, 2017).

[18] B. Fan. „Analysis on the Security Architecture of ZigBee Based on IEEE 802.15.4“. In: *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*. 2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS). Mar. 2017, pp. 241–246.

[19] N. Vidgren, K. Haataja, J. L. Patino-Andres, et al. „Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned“. In: *2013 46th Hawaii International Conference on System Sciences*. 2013 46th Hawaii International Conference on System Sciences. Jan. 2013, pp. 5132–5138.

[20] Z-Wave Alliance. *Z-Wave - Smart Home IoT Development Platform*. July 20, 2016. URL: `http://z-wave.sigmadesigns.com/` (visited on May 21, 2017).

[21] M. B. Yassein, W. Mardini, and A. Khalil. „Smart Homes Automation Using Z-Wave Protocol“. In: *2016 International Conference on Engineering MIS (ICEMIS)*. 2016 International Conference on Engineering MIS (ICEMIS). Sept. 2016, pp. 1–6.

[22] *G.9959 : Short Range Narrow-Band Digital Radiocommunication Transceivers - PHY, MAC, SAR and LLC Layer Specifications*. URL: `https://www.itu.int/rec/T-REC-G.9959-201501-I/en` (visited on May 26, 2017).

[23] ABR. *Introduction to the Z-Wave Security Ecosystem*. URL: `http://z-wave.sigmadesigns.com/wp-content/uploads/2016/08/Z-Wave-Security-White-Paper.pdf` (visited on Oct. 29, 2017).

[24] Behrang Fouladi and Sahand Ghanoun. „Security Evaluation of the Z-Wave Wireless Protocol“. In: *Black hat USA* 24 (2013), pp. 1–2.

[25] J. Picod, Arnaud Lebrun, and J. Demay. „Bringing Software Defined Radio to the Penetration Testing Community“. In: *Black Hat USA* (2014).

[26] Geoff Mulligan. „The 6LoWPAN Architecture“. In: *Proceedings of the 4th Workshop on Embedded Networked Sensors*. EmNets '07. New York, NY, USA: ACM, 2007, pp. 78–82. ISBN: 978-1-59593-694-3.

[27] P. Srivastava and Sandhya Tiwari. „An Overview of Open System Interconnection (OSI): A Seven Layered Model“. In: *Journal of Computer, Internet and Network Security*. 1.3 (Jan. 10, 2017).

[28] C. Hennebert and J. D. Santos. „Security Protocols and Privacy Issues into 6LoWPAN Stack: A Synthesis“. In: *IEEE Internet of Things Journal* 1.5 (Oct. 2014), pp. 384–398. ISSN: 2327-4662.

[29] Shahid Raza, Thiemo Voigt, and Utz Roedig. „6LoWPAN Extension for IPsec“. In: *Proceedings of the IETF-IAB International Workshop on Interconnecting Smart Objects with the Internet* (Jan. 1, 2011).

[30] S. Raza, S. Duquennoy, T. Chung, et al. „Securing Communication in 6LoWPAN with Compressed IPsec“. In: *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*. 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS). June 2011, pp. 1–8.

[31] Rene Hummen, Jens Hiller, Hanno Wirtz, et al. „6LoWPAN Fragmentation Attacks and Mitigation Mechanisms“. In: *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '13. New York, NY, USA: ACM, 2013, pp. 55–66. ISBN: 978-1-4503-1998-0.

[32] P. Pongle and G. Chavan. „A Survey: Attacks on RPL and 6LoWPAN in IoT“. In: *2015 International Conference on Pervasive Computing (ICPC)*. 2015 International Conference on Pervasive Computing (ICPC). Jan. 2015, pp. 1–6.

[33] H. S. Kim, J. Lee, and J. W. Jang. „BLEmesh: A Wireless Mesh Network Protocol for Bluetooth Low Energy Devices“. In: *2015 3rd International Conference on Future Internet of Things and Cloud*. 2015 3rd International Conference on Future Internet of Things and Cloud. Aug. 2015, pp. 558–563.

[34] *Specifications — Bluetooth Technology Website*. URL: `https://www.bluetooth.com/specifications` (visited on Oct. 22, 2017).

[35] G. Kwon, J. Kim, J. Noh, et al. „Bluetooth Low Energy Security Vulnerability and Improvement Method“. In: *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. 2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia). Oct. 2016, pp. 1–4.

[36] Xiaobing He, Michael Niedermeier, and Hermann de Meer. „Dynamic Key Management in Wireless Sensor Networks: A Survey“. In: *Journal of Network and Computer Applications* 36.2 (Mar. 1, 2013), pp. 611–622. ISSN: 1084-8045.

[37] Mohamed Eltoweissy, M. Hossain Heydari, Linda Morales, et al. „Combinatorial Optimization of Group Key Management“. In: *Journal of Network and Systems Management* 12.1 (Mar. 1, 2004), pp. 33–50. ISSN: 1064-7570, 1573-7705.

[38] Y. Zhang, Y. Shen, and S. Lee. „A Cluster-Based Group Key Management Scheme for Wireless Sensor Networks“. In: *2010 12th International Asia-Pacific Web Conference*. 2010 12th International Asia-Pacific Web Conference. Apr. 2010, pp. 386–388.

[39] W.T. Li, C.-H Ling, and Min-Shiang Hwang. „Group Rekeying in Wireless Sensor Networks: A Survey“. In: *International Journal of Network Security* 16 (Jan. 1, 2014), pp. 401–410.

[40] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. „LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks“. In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. CCS '03. New York, NY, USA: ACM, 2003, pp. 62–72. ISBN: 978-1-58113-738-5.

[41] Jing Deng, C. Hartung, R. Han, et al. „A Practical Study of Transitory Master Key Establishment ForWireless Sensor Networks“. In: *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*. First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05). Sept. 2005, pp. 289–302.

[42] Xing Zhang, Jingsha He, and Qian Wei. „EDDK: Energy-Efficient Distributed Deterministic Key Management for Wireless Sensor Networks“. In: *EURASIP Journal on Wireless Communications and Networking* 2011 (2011), pp. 1–11. ISSN: 1687-1472, 1687-1499.

[43] Y. Wang, B. Ramamurthy, and X. Zou. „KeyRev: An Efficient Key Revocation Scheme for Wireless Sensor Networks“. In: *2007 IEEE International Conference on Communications*. 2007 IEEE International Conference on Communications. June 2007, pp. 1260–1265.

[44] Kwang-Jin Paek, Ui-Sung Song, Hye-Young Kim, et al. „Energy-Efficient Key-Management (EEKM) Protocol for Large-Scale Distributed Sensor Networks.“ In: *Journal of Information Science & Engineering* 24.6 (2008).

[45] M.-L. Messai, M. Aliouat, and H. Seba. „Tree Based Protocol for Key Management in Wireless Sensor Networks“. In: *EURASIP J. Wirel. Commun. Netw.* 2010 (Apr. 2010), 59:1–59:13. ISSN: 1687-1472.

[46] C. Ma, G. Geng, H. Wang, et al. „Location-Aware and Secret Share Based Dynamic Key Management Scheme for Wireless Sensor Networks“. In: *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*. 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing. Vol. 1. Apr. 2009, pp. 770–773.

[47] Chien-Lung Wang, Tzung-Pei Hong, Gwoboa Horng, et al. „A GA-Based Key-Management Scheme in Hierarchical Wireless Sensor Networks“. In: *International Journal of Innovative Computing, Information and Control* 5 (Dec. 1, 2009).

[48] S. H. Seo, J. Won, S. Sultana, et al. „Effective Key Management in Dynamic Wireless Sensor Networks“. In: *IEEE Transactions on Information Forensics and Security* 10.2 (Feb. 2015), pp. 371–383. ISSN: 1556-6013.

[49] Ronald Watro, Derrick Kong, Sue-fen Cuti, et al. „TinyPK: Securing Sensor Networks with Public Key Technology“. In: *Proceedings of the 2Nd ACM Workshop on Security of Ad Hoc and Sensor Networks*. SASN '04. New York, NY, USA: ACM, 2004, pp. 59–64. ISBN: 978-1-58113-972-3.

[50] A. Liu and P. Ning. „TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks“. In: *2008 International Conference on Information Processing in Sensor Networks (Ipsn 2008)*. 2008 International Conference on Information Processing in Sensor Networks (Ipsn 2008). Apr. 2008, pp. 245–256.

[51] E. Fernandes, J. Jung, and A. Prakash. „Security Analysis of Emerging Smart Home Applications“. In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016 IEEE Symposium on Security and Privacy (SP). May 2016, pp. 636–654.

[52] E. Ronen, A. Shamir, A. O. Weingarten, et al. „IoT Goes Nuclear: Creating a ZigBee Chain Reaction“. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017 IEEE Symposium on Security and Privacy (SP). May 2017, pp. 195–212.

[53] Atmel. *AT15735: Atmel Smart Plug Firmware User Guide*. URL: `http://ww1.microchip.com/downloads/en/AppNotes/Atmel-42688-Atmel-Smart-Plug-Firmware-User-Guide_AT15735_ApplicationNote.pdf` (visited on Oct. 29, 2017).

[54] Jason Doyle. *Contribute to Google-Nest-Cam-Bug-Disclosures Development by Creating an Account on GitHub*. Sept. 6, 2017. URL: `https://github.com/jasondoyle/Google-Nest-Cam-Bug-Disclosures` (visited on Oct. 29, 2017).

[55] Grant Hernandez, Orlando Arias, Daniel Buentello, et al. „Smart Nest Thermostat: A Smart Spy in Your Home“. In: *Black Hat USA* (2014).

[56] Muhammad Omer Farooq and Thomas Kunz. „Operating Systems for Wireless Sensor Networks: A Survey“. In: *Sensors (Basel, Switzerland)* 11.6 (2011), pp. 5900–5930. ISSN: 1424-8220.

[57] *Contiki: The Open Source Operating System for the Internet of Things*. URL: `http://www.contiki-os.org/` (visited on Oct. 29, 2017).

[58] *Tinyos-Main: Main Development Repository for TinyOS (an OS for Embedded, Wireless Devices)*. Oct. 27, 2017. URL: `https://github.com/tinyos/tinyos-main` (visited on Oct. 29, 2017).

[59] *RIOT - The Friendly Operating System for the Internet of Things*. URL: `http://riot-os.org/` (visited on Oct. 29, 2017).

[60] P. Levis, S. Madden, J. Polastre, et al. „TinyOS: An Operating System for Sensor Networks“. In: *Ambient Intelligence*. Springer, Berlin, Heidelberg, 2005, pp. 115–148. ISBN: 978-3-540-23867-6.

[61] A. Dunkels, B. Gronvall, and T. Voigt. „Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors“. In: *29th Annual IEEE International Conference on Local Computer Networks*. 29th Annual IEEE International Conference on Local Computer Networks. Nov. 2004, pp. 455–462.

[62] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, et al. „Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems“. In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. SenSys '06. New York, NY, USA: ACM, 2006, pp. 29–42. ISBN: 978-1-59593-343-0.

[63] E. Baccelli, O. Hahm, M. Gunes, et al. „RIOT OS: Towards an OS for the Internet of Things“. In: *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Apr. 2013, pp. 79–80.

[64] M. Darianian and M. P. Michael. „Smart Home Mobile RFID-Based Internet-of-Things Systems and Services“. In: *2008 International Conference on Advanced Computer Theory and Engineering*. 2008 International Conference on Advanced Computer Theory and Engineering. Dec. 2008, pp. 116–120.

[65] *ISO/IEC 7816-3:2006 - Identification Cards – Integrated Circuit Cards – Part 3: Cards with Contacts – Electrical Interface and Transmission Protocols*. URL: `https://www.iso.org/standard/38770.html` (visited on Oct. 29, 2017).

[66] Martin Klimke and Josef Haid. *Hardware-Based Secure Identities for Machines in Smart Factories*. Infineon Technologies AG, 2016.

[67] S. Babar, A. Stango, N. Prasad, et al. „Proposed Embedded Security Framework for Internet of Things (IoT)“. In: *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE)*. 2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE). Feb. 2011, pp. 1–5.

[68] *Android*. URL: `https://www.android.com/` (visited on Oct. 29, 2017).

[69] P. Ali-Rantala, L. Sydanheimo, M. Keskilammi, et al. „Indoor Propagation Comparison between 2.45 GHz and 433 MHz Transmissions“. In: *IEEE Antennas and Propagation Society International Symposium (IEEE Cat. No.02CH37313)*. IEEE Antennas and Propagation Society International Symposium (IEEE Cat. No.02CH37313). Vol. 1. 2002, 240–243 vol.1.

[70] P. Ali-Rantala, L. Ukkonen, L. Sydanheimo, et al. „Different Kinds of Walls and Their Effect on the Attenuation of Radiowaves Indoors“. In: *IEEE Antennas and Propagation Society International Symposium. Digest. Held in Conjunction with: USNC/CNC/URSI North American Radio Sci. Meeting*. IEEE Antennas and Propagation Society International Symposium. Digest. Held in Conjunction with: USNC/CNC/URSI North American Radio Sci. Meeting. Vol. 3. June 2003, 1020–1023 vol.3.

[71] Technologies AG Infineon. *XMC4500 - Data Sheet*. URL: `https://www.infineon.com/dgdl/?fileId=5546d46254e133b40154e1b56cbe0123` (visited on Oct. 29, 2017).

[72] Technologies AG Infineon. *TDA5340 - Data Sheet*. URL: `https://www.infineon.com/dgdl/?fileId=db3a30433408410401340ea9a6a4300a` (visited on Oct. 29, 2017).

[73] Technologies AG Infineon. *SmartLEWIS TRX - TDA5340 - Product Brief*. URL: https://www.infineon.com/dgdl/?fileId=db3a3043324cae8c01325d0b47b409d0 (visited on Oct. 29, 2017).

[74] Technologies AG Infineon. *XMC1100 - Data Sheet*. URL: https://www.infineon.com/dgdl/?fileId=5546d46255dd933d0155e31763e577dc (visited on Oct. 29, 2017).

[75] Technologies AG Infineon. *TLI4970-D025T4 - Data Sheet*. URL: https://www.infineon.com/dgdl/?fileId=5546d4625607bd1301562bdf09d8339f (visited on Oct. 29, 2017).

[76] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - AES - The Advanced Encryption Standard*. 1. Springer-Verlag Berlin Heidelberg. 238 pp. ISBN: 978-3-662-04722-4.

[77] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005. ISBN: 978-0-13-186239-5.

[78] Elaine Baker. *Suite B Cryptography*. March, 2006. URL: https://csrc.nist.gov/CSRC/media/Events/ISPAB-MARCH-2006-MEETING/documents/E_Barker-March2006-ISPAB.pdf (visited on Aug. 22, 2017).

[79] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, Nov. 6, 2014. 598 pp. ISBN: 978-1-4665-7027-6.

[80] Daniel R. L Brown. „Elliptic Curve Cryptography". In: *Standards for Efficient Cryptography 1 (SEC 1)* 2.0 (May 2009).

[81] V. Gayoso Martinez, L. Hernandez Encinas, and C. Sanchez Avila. „A Java Implementation of the Elliptic Curve Integrated Encryption Scheme". In: *The 2010 International Conference on Security and Management (SAM'10)*. 2010.

[82] Technologies AG Infineon. *Infineon Technologies - DAVE Development Platform*. URL: https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/ (visited on Nov. 6, 2017).

[83] Eclipse Foundation Inc. *Eclipse - The Eclipse Foundation Open Source Community Website*. URL: https://www.eclipse.org/home/index.php (visited on Nov. 6, 2017).

[84] Android. *Android Studio and SDK Tools*. URL: https://developer.android.com/studio/index.html (visited on Nov. 2, 2017).

[85] Technologies AG Infineon. *XMC4500 Relax Kit - Board User Manual*. URL: https://www.infineon.com/dgdl/?fileId=db3a30433acf32c9013adf6b97b112f9 (visited on Oct. 29, 2017).