Thomas Ulz, BSc.

Design and Implementation of an NFC-based Configuration Interface for Smart Sensors

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Institute for Technical Informatics

Advisors

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Dipl.-Ing. Holger Bock, Infineon Technologies Austria AG

Graz, September 2016

Abstract

The fourth industrial revolution, named *Industry 4.0* by the German government, introduces industrial cyber-physical systems. Unlike traditional manufacturing devices that were kept isolated from the Internet, the goal of Industry 4.0 is to connect each and every device that is involved in the production processes to the Internet. This allows for machine-to-machine and machine-to-human communication which can be used to monitor, control, optimize and change production flows. By engaging in the production flow, highly customized products manufactured to customers' needs instead of mass produced items can be crafted.

In the context of this Industry 4.0 movement, the *IoSense*[1] project, in which scope this master's thesis was conducted, tries to develop a *Flexible Frontend/Backend Sensor Pilot Line for the Internet of Everything*. The smart sensors, produced as part of such a sensor pilot line, can be used in a variety of domains, such as in smart factories or smart homes. Independent of the context in which such a smart sensor will be utilized, by enabling configuration changes throughout the sensor's whole life cycle, production of the sensor becomes more flexible. However, if such a configuration interface is integrated into a smart sensor, security measures need to be taken as potential adversaries could either read confidential configuration parameters or damage the hardware by applying a malicious configuration. Furthermore, energy consumption of the configuration interface must be considered as smart sensors might not be powered during production where initial configurations are applied. Also, if a smart sensor is operated wireless, energy efficiency is of utmost importance.

Thus, in this master's thesis, an NFC-enabled configuration interface for Smart Sensors is presented. By using NFC technology, the hardware necessary to process configurations can be powered using the NFC field. To secure the configuration interface and the transported data, security measures are analyzed in this thesis and applied to the implemented prototype. The evaluation of this prototype includes a threat analysis which demonstrates the countermeasures taken concerning various threats. Also, it is shown that the overhead imposed by applying the implemented security measures is reasonable for realistic configuration sizes of about 1 kB.

[1] *IoSense* is an EU project funded by ECSEL

Kurzfassung

Die vierte industrielle Revolution, von der deutschen Regierung auch als Industrie 4.0 bezeichnet, führt zu industriell genutzten cyber-physischen Systemen. Anders als bei traditionellen Produktionsmaschinen, welche oft nicht mit dem Netzwerk verbunden wurden, ist das Ziel von Industrie 4.0 jede im Produktionsprozess involvierte Maschine mit dem Internet zu verbinden. Das ermöglicht Maschine-zu-Maschine sowie Maschine-zu-Mensch Kommunikation, welche genutzt werden kann, um den Produktionsprozess zu überwachen, regeln, optimieren und zu ändern. Durch das Eingreifen in den Produktionsprozess wird das Erzeugen maßgeschneiderter Produkte entsprechend von Kundenanforderungen anstelle von massenproduzierten Artikeln ermöglicht.

Im Kontext dieser Industrie 4.0 Bewegung versucht das EU-Projekt *IoSense*[2], im Rahmen dessen diese Maserarbeit erstellt wurde, eine flexible Front-End/Back-End Sensor Pilotlinie für das Internet der Dinge zu entwickeln. Die intelligenten Sensoren, welche im Zuge solch einer Pilotlinie entwickelt werden, können in einer Vielzahl von Anwendungsgebieten wie intelligenten Fabriken oder intelligenten Häusern eingesetzt werden. Unabhängig vom Einsatzgebiet des Sensors sollte dieser über seinen gesamten Lebenszyklus konfigurierbar sein, um die Produktion flexibler gestalten zu können. Wenn aber eine Konfigurationsschnittstelle in intelligente Sensoren integriert wird, muss auch die Sicherheit der Daten berücksichtigt werden. Angreifer könnten diese Schnittstelle nutzen, um Informationen auszulesen oder den Sensor zu beschädigen. Zusätzlich muss auch der Energieverbrauch solch einer Konfigurationsschnittstelle bedacht werden, da die Sensoren zumindest während der Produktionsphase nicht mit Energie versorgt werden können. Wird der Sensor in einer kabellosen Umgebung eingesetzt, muss Energieeffizienz ohnehin als sehr wichtiger Punkt betrachtet werden.

Daher wird in dieser Masterarbeit eine NFC-basierte Konfigurationsschnittstelle für intelligente Sensoren vorgestellt. Durch die Benützung von NFC kann die Konfigurationsschnittstelle durch das NFC-Feld mit Energie versorgt werden. Um die Schnittstelle abzusichern, werden geeignete Sicherheitsmaßnahmen implementiert. Der gezeigte Prototyp wird anhand einer Sicherheitsanalyse evaluiert. Zusätzlich zeigt die Evaluierung, dass durch die implementierten Sicherheitsmaßnahmen ein vertretbarer Mehraufwand bei der Datenübertragung von realistischen Paketen mit einer Größe von 1 kB entsteht.

[2] *IoSense* ist ein EU Projekt gefördert durch ECSEL

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 master's thesis dissertation.

Date	Signature

Acknowledgements

At this place, I want to thank all people that have helped, encouraged, guided and advised me throughout this master's thesis and my whole studies. First, I want to thank Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger from the Institute for Technical Informatics, as well as Dipl.-Ing. Holger Bock and Dipl.-Ing. Thomas Ruprechter, both from Infineon Austria, for providing me the opportunity to work in such an interesting and promising field during my thesis. I also want to thank them for their inputs and guidance while working on this master's thesis.

I also want to thank all colleagues at Infineon for their support, feedback and ideas regarding the concept presented in this work. In particular, I want to thank Andreas Wallner, MSc and Dipl.-Ing. Dr.techn. Christian Lesjak for introducing me to relevant concepts and for providing me with valuable input and feedback regarding my work.

Finally, I want to thank my family and friends for their immense support during this master's thesis and my whole studies. Without them, achieving all goals set during the last years would not have been possible. In particular, I want to thank my girlfriend Sarah, my parents Renate and Johann, and my friends and colleagues Christian, Jakob, Jürgen, Marco, Michael, and Thomas.

List of Figures

List of Tables

Contents

1 Introduction

This introductory chapter comprises several parts. First, a motivation for the chosen topic is given, where the need for the proposed solution will be expressed. In the second part, an introduction and overview of the existing system that is going to be extended through the work of this master's thesis is given. After that, the goals that were defined for this thesis are listed. The chapter is concluded with an overview of the rest of this document's contents.

1.1 Motivation

Today's economic pressure causes the industry to offer personalized products, specially manufactured to each customer's needs. This is in contrast to mass production, where the goal is to produce as many products at costs as low as possible. Another trend in manufacturing calls for resource and energy efficient production, as environmental changes arise. Those two mentioned aspects led to a high-tech strategy by the German government, which coined the term Industry 4.0. The changes proposed and predicted by this initiative are also seen as the fourth industrial revolution. An overview of all four of those industrial revolutions is depicted in Figure 1.1.

The first industrial revolution, taking place in the late 18th century, was driven by the usage of steam and water powered machines which led to mechanization of production processes. By using electricity, assembly lines and therefore mass production became possible in the second industrial revolution which took place at the beginning of the 20th century. In the early 1970s, automation and the use of computing systems led to the third industrial revolution. The fourth industrial revolution will be driven by connecting

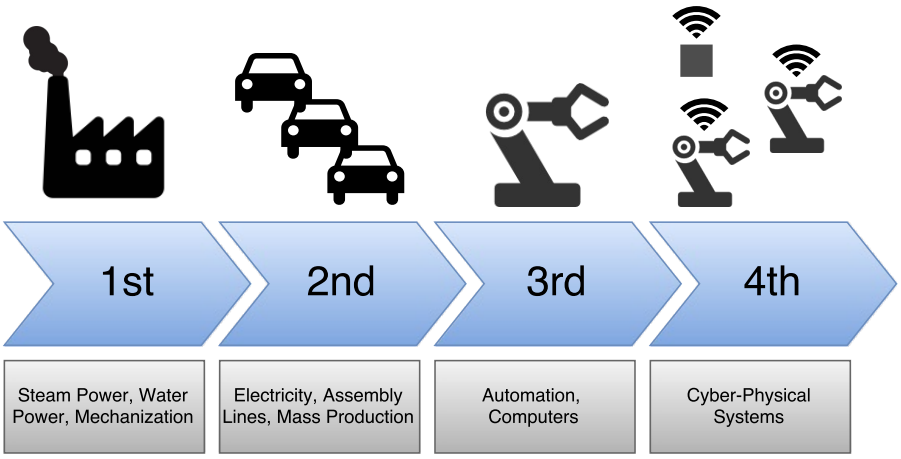


Figure 1.1: Overview of all four industrial revolutions.

factory equipment to the internet, thus making them *cyber-physical systems*. In addition to potentially existing production equipment, also the addition of (wireless) sensors is envisioned in the Industry 4.0 initiative, thus, including factories into the Internet of Things.

By connecting factory equipment and sensors to the Internet, new problems arise which include making machines vulnerable to attacks from arbitrary adversaries. Attacks could have multiple effects like denial of service or espionage from competitors. Thus, research is currently trying to find security solutions. Those solutions, amongst others, require integration of special hardware to make production equipment secure. However, also the configuration of this security related hardware needs to be done in a secure way.

To enable (on site) configuration of the equipment and sensors even before enabling any remote connectivity, an NFC-based configuration solution will be proposed in this work. The focus of this thesis is the security of configuration data, which might include confidential data such as key material, as well as on the practicability and ease of configuration. By making the configuration process fail safe as well as secure, any maintenance worker will be able to configure equipment.

Even before the term *Industry 4.0* was coined, Koren et al. [KHJ+99] introduced the term *Reconfigurable Manufacturing System (RMS)* which is a vital concept in the Industry 4.0 context. Koren et al. [KHW98] also point out that configuring a manufacturing system plays an important role in impacting the systems' performance. Lee et al. [LBK15] propose a 5-level structure for the development of cyber-physical systems which are used in the manufacturing process. Among others, the authors highlight configuration as one of the five proposed levels as shown in Figure 1.2.

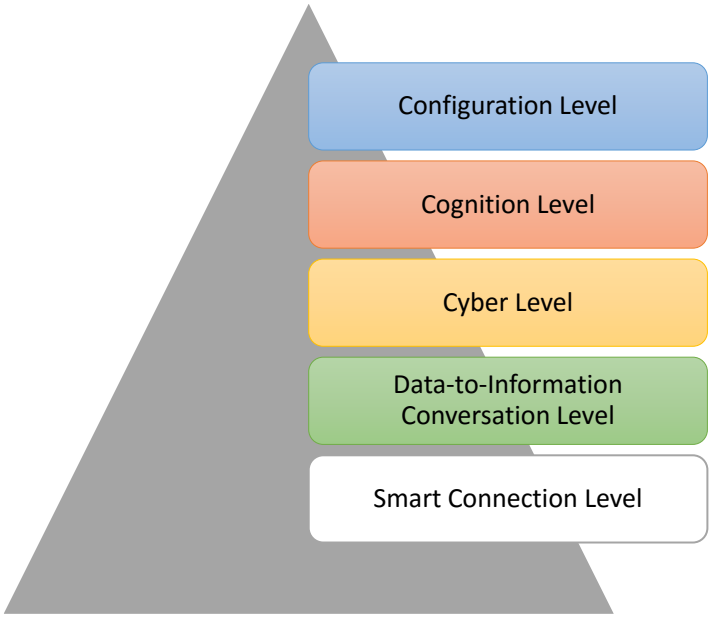


Figure 1.2: Five level structure for the development of cyber physical systems which are used in the manufacturing process as proposed by Lee et al. [LBK15].

1.2 Introduction

The core idea of this master's thesis is to use the work done by Lesjak et al. [LRH+14], [LRB+14a], [LRB+14b], [LHH+15], [LHW15] as a basis architecture which will be extended by a secure configuration capability. The motivation for the work done by Lesjak et al. is to enable *smart maintenance services* for existing equipment. The aim of smart maintenance services is to optimize work-intensive maintenance, repair, and operations (MRO) tasks. This is done by collecting data from industrial equipment to anticipate service needs.

The main hardware component, denoted as *mediator* in the work of Lesjak et al., can be applied to existing (legacy) factory equipment. A basic overview of its components is depicted in Figure 1.3. There, various connection interfaces can be seen. The *Equipment Interface* is used to connect the mediator's *Equipment Host Controller* to any existing production equipment. Here, a range of physical connectors and protocols such as Ethernet, USB, serial connection, CAN bus, Profibus, and EtherCAT are supported. These interfaces are used to collect equipment *health and condition* information, further referred to as equipment *Snapshot*.

The *Network Interface* will then be used to transmit the collected snapshots to so-called *smart service backends*, provided by the equipment vendors. Because those backends will not be located within the same facility as the hardware the mediator is deployed to,

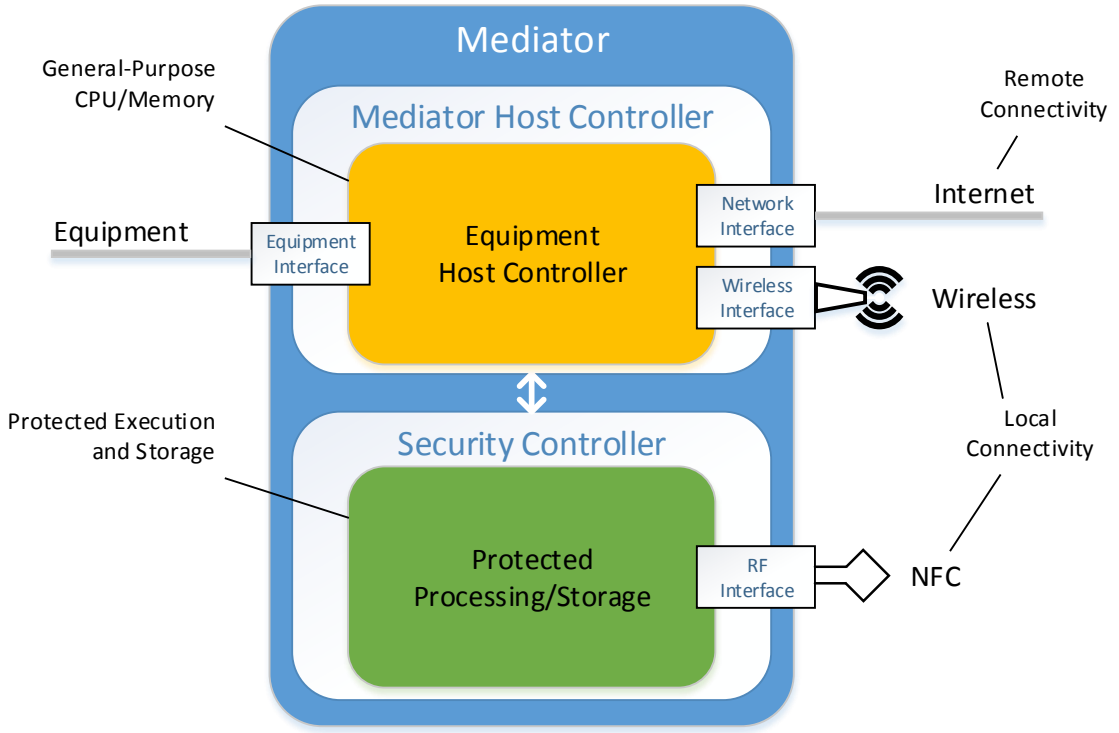


Figure 1.3: Architecture overview of mediator hardware element.

the *Network Interface* needs to support the Internet Protocol (IP) to enable connections to the Internet.

Local Connectivity is provided by two interfaces. The *Wireless Interface* can be used to connect to the mediator via a wireless local area network (WLAN). The *RF Interface* offers a near field communication (NFC) interface. Both interfaces for local connectivity can be used by maintenance technicians to access the mediator.

As can be seen in the architectural overview depicted in Figure 1.3, the mediator comprises a Host Controller as well as a Security Controller (SC). The SC provides a number of security related features, which will be discussed in Section 2.1.1. Because of its secure nature, the following functions are provided by the SC:

- Secure storage for software and data (includes cryptographic credentials)
- Secure execution of code
- Cryptographically secure random number generator

By using the two mentioned controllers, the mediator's storage and processing capabilities are split. Communication and the respective protocols are handled on the powerful

general-purpose Host Controller, while the Secure Controller is responsible for providing a Secure Execution Environment (SEE). All components necessary for implementing a secure NFC-based configuration interface which can be attached to (legacy) equipment exist in the mediator hardware.

- The Security Controller's NFC interface is used for transmitting configurations to the mediator.
- The Security Controller is used to perform decryption and verification of transmitted configuration packages. Also, key material is stored inside the security controller's protected storage.
- Also, other confidential configuration parameters can be stored in the security controller's protected storage.
- The Host Controller and its equipment interface are used to propagate the configuration to the attached manufacturing devices.

Throughout the rest of this thesis, the utilized mediator hardware will be denoted as *secure element* to use a more generalized naming.

1.3 Goals

The main goal of this master's thesis is to design and implement an NFC-based configuration interface. The requirements for that interface are defined in the following list.

Security: Key material and other configuration parameters should be secured in any step of the configuration process. It must not be possible for adversaries to obtain, read or manipulate the transported data.

Usability: The implemented software should be easily usable for maintenance workers. The software should assist the workers in a way such that user errors are limited to a minimal amount.

Portability: The two components (mobile device and backend) needed for the configuration process besides the previously mentioned mediator should not require any special hardware other than an NFC interface. It should be possible to use components off the shelf for all involved parts of the system (of course besides the mediator).

1.4 Overview

The rest of this work will be structured as follows. In Chapter 2 - *Prerequisites and Related Work*, the technical aspects of this master's thesis will be discussed. All involved technologies such as NFC, the secure element as well as the necessary cryptographic principles will be discussed there. Also in this chapter, related work to the topics covered in this master's thesis will be given. After that, the design process will be documented in Chapter 3 - *Design*. The implementation, as well as the used tools and libraries will be discussed in Chapter 4 - *Implementation*. To evaluate the implemented system, a threat and risk analysis was conducted. The results as well as evaluations with respect to data size will be presented in Chapter 5 - *System Evaluation*. In Chapter 6 - *Conclusion and Future Work*, known limitations of the implemented solution as well as possible future work are listed. This master's thesis then will be completed by a conclusion given in that same chapter.

2 Prerequisites and Related Work

In this chapter, several components and technologies which were used in the implementation of this master's thesis are going to be discussed. Also, related work for the topics involved in this thesis is listed and reviewed. To finish the chapter, differences of the implementation presented in this thesis to comparable, state of the art solutions are highlighted.

2.1 Prerequisites

The (technical) background for the components and technologies that is given in this section is essential for understanding the concepts and implementation presented in Chapters 3 and 4.

2.1.1 Security Controller (SC)

Before discussing Security Controllers in detail, the fundamental concept of *Security by Isolation* will be discussed.

Security by Isolation

Vasudevan et al. [VOZ+12] propose to split an execution environment into a *normal world* and a *secure world*. The normal world, according to the authors, would represent a general-purpose execution environment (GPEE) while the secure world would serve as

a secure execution environment (SEE). The same concept was adapted in many publications and is also known as dual-execution [SAB15], or red and green worlds [LHW15]. Vasudevan et al. [VOZ+12] identify five security features that enable secure execution for mobile devices:

Isolated Execution: offers the possibility for application developers to run their code completely isolated from other code.

Secure Storage: is providing secrecy, integrity and/or freshness for data. This applies especially when the device is powered off, but also under certain conditions depending on which software currently is loaded.

Remote Attestation: allows remote parties the verification if a particular message originates from a particular software module.

Secure Provisioning: allows sending data to a specific software module running on a specific device. The integrity and secrecy of that data are guaranteed.

Trusted Path: protects the communication between a software module and a peripheral (authenticity and optionally secrecy and availability).

Security Controllers

Security Controllers might store or operate with cryptographic key material or other sensitive data. Therefore, they become an attractive target for attackers. Anderson et al. [ABCS06] list four types of attacks against which security controllers should be resistant.

Invasive attacks: entail access to the (electrical) internal components of a crypto processor. Attackers might, for instance, remove the packaging, drill a hole into a chip and probe signals on bus lines.

Semi-invasive attacks: do not damage the hardware, but still, involve physical access to it. Attackers might try to change internal states by, for instance, trying to alter the state of a flip-flop by inducing power spikes.

Local non-invasive attacks: involve detailed observation of device parameters such as power consumption. In such an attack, also called side-channel attack, an adversary might precisely measure the power consumption and correlate the trace to the data processed.

Remove attacks: include the observation of the device's standard inputs and outputs. Potential attacking points include timing analysis, the analysis of protocols or the exploitation of programming interfaces. For instance, timing measures can be used to correlate the traces to processed data. This particular attack also falls into the category of side-channel attacks.

To provide security by isolation, security controllers are conducted as external hardware modules. However, as only the SEE can be realized on such security controllers, also communication interfaces to the GPEE will be necessary. Therefore, current SCs offer interfaces such as USB, I2C or SPI. Also, contactless interfaces such as NFC can be integrated into security controllers.

2.1.2 NFC

Near Field Communication is a wireless communication technique, based on several RFID (radio-frequency identification) standards. It operates at a radio frequency of 13.56 MHz, up to a range of approximately 10 cm. Because the NFC standard comprises various RFID standards, NFC devices are compatible with existing RFID cards and tags, as Gauthier Van Damme and Karel Wouters [VDWP09] note. The NFC standard ISO/IEC 18092 or ECMA-340 [fSEC+04] called *Near Field Communication Interface and Protocol-1 (NFCIP-1)* defines the NFC interface and communication protocols. Also in this standard, three supported bit rates are defined: 106, 212 and 424 kbits per second (kbps).

As mentioned by Roy Want [Wan11], the standard furthermore defines different operation modes for NFC. According to the standard, devices can communicate in a passive or an active way. In the passive mode, one device, the initiator, acts as a reader and powers the passive target device. The initiator powers the target by generating an RF field which is modulated by the passive device. In active mode, both devices are powered and thus, can generate their own RF field. Later, a second standard ISO/IEC 21481 or ECMA-352 [fSEC+05] also called *NFCIP-2* defined three standard operation modes for NFC devices:

Card Emulation Mode: In this mode, the NFC device emulates a (smart) card. No RF field is generated by the device; it is operated in passive mode.

Reader/Writer Mode: The NFC device generates an RF field to communicate with a passive device (smart card, RFID tags). The device, therefore, acts like a normal active contactless card reader.

Peer to Peer Mode: To communicate actively with a second NFC device, this mode is used. A master/slave principle is applied where the master (initiator) starts the data transfer. After that it waits for the slave (target) to respond. The two devices can communicate with each other both in active or passive NFC mode.

A lot of research regarding the security of NFC was already made in the META[:SEC:] project by Druml et al. [DMK+13], [DMK+14], and Höller et al. [HDK+14]. Some principles and associated attacks will be discussed in the following section.

2.1.3 Security of NFC

To better understand why some of the design decisions in Chapter 3 were made, the security issues of NFC are highlighted here. The most common (possible) attacks for NFC communication will be discussed here. The categorization of attacks was also discussed by Ernst Haselsteiner and Klemens Breitfuß [HB06].

Eavesdropping

The communication between devices in NFC is based on RF waves, as mentioned previously. Therefore, eavesdropping is an obvious issue for NFC. Attackers can use antennas and analysis equipment to listen to any data being sent between two devices. However, as discussed above, the usual communication range for NFC devices is approximately 10 cm. This close proximity implies a low power RF field, which makes NFC communication harder to attack than other wireless protocols. The distance an attacker needs to be within to successfully eavesdrop NFC communication actually depends on a number of parameters such as the quality and characteristics of the attackers RF antenna and receiver, the setup of the location where the attack would be performed or the power sent out by the transmitting NFC device. As stated by Haselsteiner and Breitfuß [HB06], approximately 10 m for active and 1 m for passive devices should be considered as a rule of thumb for possible eavesdropping.

Data Manipulation

For data manipulation, three different scenarios are distinguished:

Data Corruption: If an attacker is not interested in the transmitted data but instead just wants to disturb the communication, she needs to transmit data for valid frequencies at the correct time. If the modulation scheme and coding are known, it is easy for the attacker to calculate these correct times. This attack can be seen as a Denial of Service (DoS) attack.

Data Modification: If the attacker wants the receiving device to receive valid, but manipulated data, different steps for data corruption depending on the modulation scheme need to be taken. One such task, filling a pause, easily can be done by sending a signal. This changes a zero to a one. To change a one to a zero, a signal level needs to be reduced, which can be achieved by sending overlapping signals. As mentioned, depending on the chosen modulation and encoding, this attack is either feasible for certain bits and impossible for the other bits, or feasible for all transmitted bits.

Data Insertion: In this scenario, an attacker tries to insert messages into an ongoing data exchange between two devices. For this to happen, the answering device needs to take a long time to answer. In the timeslot where no data is transmitted because

the answering device needs much time, the attacker can send his own data. The attacker's data transfer, however, needs to be finished before the original answer is transmitted. If not, the two RF signals overlap and the data will be corrupted.

Man-in-the-middle

In a man-in-the-middle (MITM) attack, two devices *A* and *B* want to communicate with each other. Without their knowledge, a third party, the attacker *E*, places itself between those two parties and communicates with both. So instead of *A* sending data to *B*, *A* actually is sending data to *E*, who then forwards the information to *B*. As both parties do not know of the man-in-the-middle, *E* can observe or even manipulate the data between *A* and *B*. For the attacker, it is important that while receiving a message from, for instance, *A*, she makes sure that *B* does not see that same message but rather the message sent by *E*. The attacker, therefore, needs to jam the initial signal and send its own signal instead. NFC however, allows a device to receive and transmit data at the same time. Therefore, the sender would recognize the attacker's jamming and immediately terminate the data transmission. Additionally, NFC is a short-range communication technology. Therefore MITM attacks on NFC are practically impossible to mount.

Concluding their analysis, Gauthier Van Damme and Karel Wouters [VDWP09] state:

> *We can conclude that even if the NFC standard foresees some features that makes the attacker's life harder, perfect security can only be obtained when dedicated cryptography is used to establish a secure channel between communicating devices.*

In [PFT+14], Plosz et al. analyze different wireless communication technologies concerning security vulnerabilities in industrial usage. When discussing possible security mechanisms defined in the NFC standard, the authors note that many attacks are still possible. Therefore, in Section 2.1.5, cryptographic principles used in the implementation done for this master's thesis are going to be discussed.

2.1.4 QR Codes

A Quick Response (QR) code is a two-dimensional code (matrix barcode) which was presented by a subsidiary company of Toyota in 1994 as noted by Tan Jin Soon [Soo08]. In many areas, QR codes offer the same features as the well-known barcode technology. However, it has some major advantages when compared to linear barcodes such as a much higher data density. In addition to that, QR codes can be read from arbitrary angles. Depending on the chosen error correction, arbitrary data of up to 2,953 Bytes can be stored in a QR code.

2.1.5 Authenticated Encryption

Within the context of this master's thesis, *authenticated encryption* is used to provide *confidentiality, integrity* and *authenticity* of data. There exist specialized authenticated encryption modes for symmetric block ciphers, however, in general, authenticated encryption can be constructed by combining a encryption scheme and a *Message Authentication Code (MAC).* Therefore, before further discussing authenticated encryption, *symmetric cryptography* and *MACs* are going to be discussed.

Symmetric cryptography

Bellare et al. [BDJR97] state the main difference between symmetric (*private-key*) and asymmetric (*public-key*) cryptography:

> *An encryption scheme enables Alice to send a message to Bob in such a way that an adversary Eve does not gain significant information about the message content. This is the classical problem of cryptography. It is usually considered in one of two settings. In the symmetric (privatekey) one, encryption and decryption are performed under a key shared by the sender and receiver. In the asymmetric (public-key) setting the sender has some public information and the receiver holds some corresponding secret information.*

In this master's thesis, private-key encryption is used to secure the transferred data. One of the most widely used algorithms in that category is AES (*Advanced Encryption Standard*). According to Lu and Tseng [LT02] AES can be used with key lengths of 128, 192 or 256 bit. The authors also note that AES is a block cipher, which means that the algorithm processes plaintext blocks of 128 bits (16 bytes). Therefore, each block can be represented by a 4x4 matrix. If less than those 128 bit need to be encrypted, the plaintext will be padded with zeros. AES comprises four different steps, which, depending on the key length, are repeated for a certain number of rounds. Those four steps involve:

- **SubBytes:** Each byte is replaced by another one based on a lookup table.
- **ShiftRows:** The data in each row is shifted cyclically by a certain offset. For AES, the first row is unchanged; the second row is shifted by one, the third row by two and the last row by an offset of three.
- **MixColumns:** A mixing operation is applied to each of the four columns of the matrix.
- **AddRoundkey:** The subkey of the respective round is combined with the already calculated intermediate result.

For an AES key with a length of 256 bit, those four steps are repeated 14 times. The National Institute of Standards and Technology announced AES as being secure even for confidential government data [Hat03]:

> *The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths.*

Message Authentication Code

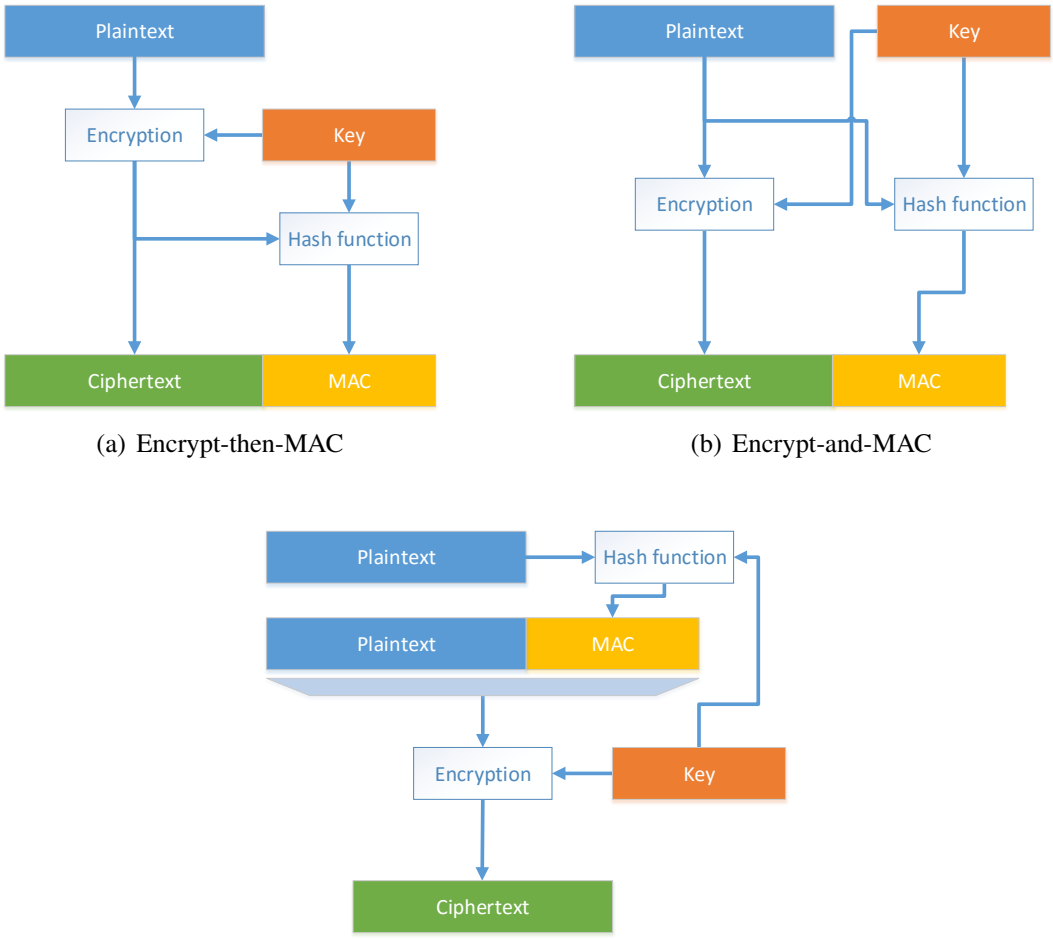
Message Authentication Codes (MACs) often are based on *hash functions* which must provide the following characteristics: *collision resistance, preimage resistance*, *and second-preimage resistance* [RS04].

- **preimage resistance**: infeasible to find x' with $y = h(x')$ for any given y.
- **2nd-preimage resistance**: infeasible to find $x' \neq x$ s.t. $h(x') = h(x)$ for given x.
- **collision resistance**: infeasible to find x, x' such that $h(x) = h(x')$.

The purpose of MACs is to ensure that a message came from a given sender (*authenticity*) and that the message was not altered by a third party (*integrity*). In contrast to a simple hash function, a MAC needs a secret key as well as the message as its input to produce the MAC value. As will be discussed in Chapter 3, an HMAC (keyed-hash message authentication code) [KCB97], based on SHA-256 will be implemented in this master's thesis.

Authenticated Encryption Modes of Operation

As Bellare and Namprempre [BN00] state, there are different ways to combine encryption and MAC to achieve authenticated encryption. The three modes, *Encrypt-then-MAC*, *Encrypt-and-MAC*, and *MAC-then-Encrypt* differ in the order of execution and the input for the encryption and MAC parts. The respective execution time point can be derived from the name of those concepts. For instance, in *Encrypt-then-MAC*, the plaintext is first encrypted using a private key. The resulting cyphertext and the private key are then used to calculate the MAC. Figure 2.1 depicts the different modes of operation for authenticated encryption.



(a) Encrypt-then-MAC

(b) Encrypt-and-MAC

(c) MAC-then-Encrypt

Figure 2.1: Authenticated Encryption modes of operation.

2.1.6 Android

Android is an operating system for mobile devices, which is developed by Google. The system is based on the Linux kernel and therefore can provide security features similar to a Linux system. Because the configuration of devices via NFC in this master's thesis will be done using an Android device, application and data security of Android need to be considered. Enck et al. conducted studies about Android security in general [EOM+09] and application security in particular [EOMC11]. The authors state, that each Android application in general runs under their unique user identity, which allows the systems to limit potential damage to single applications. The second mechanism to protect applications and data is by securing ICC (inter-component communication). This is done by mediating ICC, which, for instance, enforces a mandatory access control (MAC) on ICC. The simplest supported form of MAC is by using permissions for applications.

2.2 Related Work

In this section, related work to various topics, comprised in the implementation of this master's thesis, is given. The related work includes publications concerning Industry 4.0, the Internet of Things, smart homes, an overview of possible applications of NFC, an introduction to the concepts of Bring Your Own Device/Key as well as different configuration scenarios using NFC and QR codes.

2.2.1 Internet of Things

The term *ubiquitous computing* was first coined by Marc Weiser in 1991 [Wei91] when he stated:

> *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. [...] Silicon-based information technology, in contrast, is far from having become part of the environment. More than 50 million personal computers have been sold, and nonetheless the computer remains largely in a world of its own.*

In this work, the concept of embedding computers into things used in everyday life is introduced. Weiser forecasted, that, if the size of computers can be decreased enough, people will use hundreds of computers instead of a single *personal computer*.

> *Hundreds of computers in a room could seem intimidating at first, just as hundreds of volts coursing through wires in the walls did at one time. But like the wires in the walls, these hundreds of computers will come to be invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks.*

Based on this vision, the *Internet of Things* is a more recent interpretation of ubiquitous computing. The corresponding vision, as discussed by Friedemann Mattern and Christian Floerkemeier [MF10], foresees the extension of the Internet into the real world, everyday objects. The authors further state that physical devices, if connected to the Internet, can be remotely controlled or observed via the Internet, which is offering new opportunities for individuals and the economy. Mattern and Floerkemeier also define the term *smart objects*, which they note will play a huge role in the Internet of Things. In their definition, smart objects will be using sensors to perceive the environment they are integrated into. This information can then be communicated using the networking capabilities.

Another term used in the context of Internet of Things are *Cyber-Physical Systems (CPS)* which are discussed, for example, by Edward Lee [Lee08]. The author defines a cyber-physical system as the integration of computation into a physical process. Embedded computers with network capabilities then monitor and even control the physical

process. Lee postulates, that in order to use the full potential of CPS, new networking and computing abstractions which consider physical dynamics need to be defined.

In their survey paper, Miorandi et al. [MSDPC12] state that in 2012, 2 billions of people were using the Internet. However, the authors envision a huge jump in the number of Internet users, as machines and smart objects start to communicate via the Internet as well. An estimation by the National Cable & Telecommunications Association[3] which is shown in Figure 2.2 forecasts up to 50 billions of devices connected to the Internet by 2020.

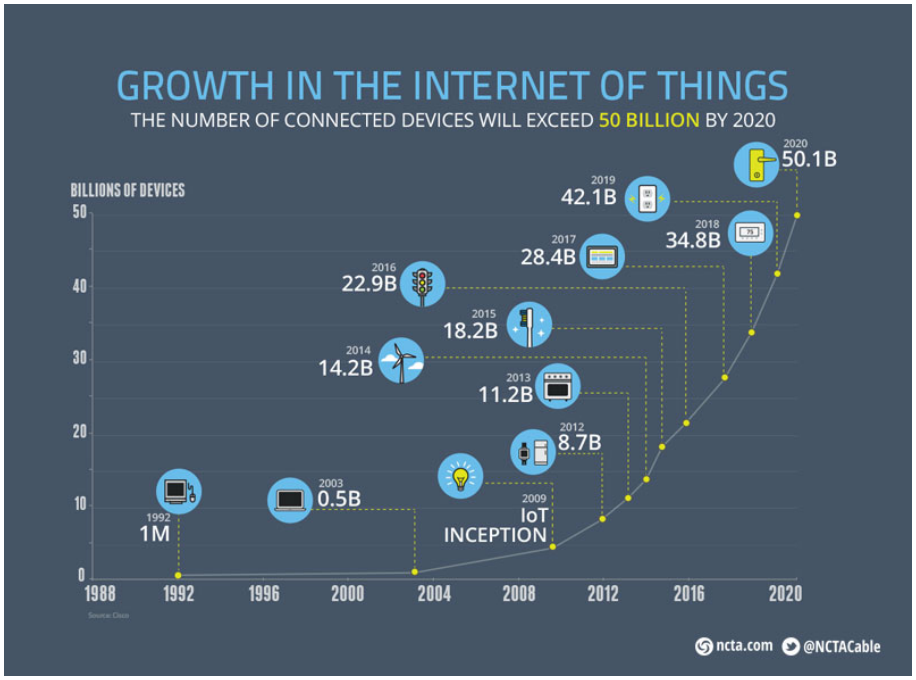


Figure 2.2: Estimated growth of the number of connected devices by the National Cable & Telecommunications Association.

As the number of devices connected to the Internet will increase rapidly, so will the amount of data that is stored, processed and transported. This larger amount of devices, services and data, however, will make attacking the Internet of Things even more interesting for adversaries. Furthermore, as Jing et al. [JVW+14] state, IoT introduces even more security problems than the traditional Internet did:

> *IoT not only has the same security issues as sensor networks, mobile communications networks and the Internet, but also has its specialties such as privacy issues, different authentication and access control network configuration issues, information storage and management and so on.*

[3] https://www.ncta.com/

Roman et al. [RNL11] analyze the current state of the art in security for the Internet of Things. When discussing technologies to secure IoT at the protocol and network level, the authors postulate that *traditional public-key infrastructures will not scale to the large number of IoT's contexts and devices*. The authors also state that in addition to security aspects such as *privacy* or *data integrity*, *fault tolerance* will become more important for IoT devices. Since billions of devices will produce and consume data, many of them being highly constraint, finding one weak link to attack might be easy for adversaries. Being in control of a single device could then lead to failures of many services.

2.2.2 Industrial IoT and Industry 4.0

Karl Steinbuch, a german computer science pioneer, stated in 1966 [Ste66]:

> *In a few decades time, computers will be interwoven into almost every industrial product.*

That vision is becoming reality nowadays as a fourth industrial revolution is said to take place. The term *Industrie 4.0* was coined by the German government in 2011 and is also known as the fourth industrial revolution. Since then, much research has been done in that particular field, especially in the context of Internet of Things which is a major component of the Industry 4.0 concept.

According to Nasser Jazdi [Jaz14], digital factories as envisioned for Industry 4.0 are characterized by the following features:

- **Smart networking:** Equipment such as production equipment, sensors, and actuators are constantly connected, either wireless or through wires.
- **Mobility:** Mobile devices such as tablets or smartphones can be used to access production relevant information and thus lead to easier diagnosis, maintenance and operation.
- **Flexibility:** Flexibility is a major feature of Industry 4.0 as production equipment as well as maintenance and operation services can be combined from various manufacturers. By using *big data*, automation can be further assisted.
- **Integration of customers:** Personalization of products will be provided because of the *flexibility* offered by Industry 4.0.
- **New innovative business models:** As mentioned, flexible and also distributed production will become possible in the future. Also, products will be personalized and modular. Thus, new and innovative business models will result from the Industry 4.0 initiative.

Jazdi also notes that safety and security will become a major concern for smart factories:

> *Industry 4.0 brings many challenges that need to be extensively studied in the research. Many questions arise: How can the reliability and safety of these products, whose development is distributed, be determined and how are they certified? Another important task is the subject of data protection and security. It must be ensured that one's own know-how and privacy are protected and remain unaffected. To this end, new concepts and technologies that allow a trustworthy cooperation of many groups and units are needed.*

Sadegi et al. [SWW15] state that industrial IoT is producing a significant amount of sensitive data which makes them attractive targets of cyber attacks. As the authors note, cyber attacks on industrial IoT devices are very critical because such attacks could cause physical damage to production equipment or even threaten human lives. Thus, Sadegi et al. give a security analysis as well as an outlook for possible solutions regarding a security framework for industrial IoT systems.

As a possible solution for cyber attacks, John Stankovic [Sta14] proposes to use self-healing mechanisms for (production) critical systems. The self-healing process involves the detection of an attack, analysis of the attack, and the deployment of countermeasures and repairs regarding the diagnosed attack. A major challenge regarding self-healing, according to the author, are resource constraint devices which require that all of the steps mentioned above need to run in a light-weight manner. One possible solution, the downloading of new code onto the devices, is presented by Deng et al. [DHM06]. However, this method is said to be insufficient by Stankovic. He makes the point that downloading and applying new code itself is prone to security-related attacks. Finally, Stankovic states:

> *It is likely that significant hardware support [RRC04] will be necessary for providing encryption, authentication, attestation, and tamper proof keys. Even if new devices are security-aware, dealing with legacy devices will prove difficult.*

Ning et al. [NLY13] list different attack categories for which the authors state various types of attacks, the possible consequences of such attacks and potential countermeasures. The list of attacks can be seen in Table 2.1.

Claudia Eckert and Niels Fallenbeck [EF15] discuss the utilization of (open) cloud services in the context of Industry 4.0. The authors highlight the importance of data protection starting at the point of data acquisition (sensors) via various transport channels up to the cloud service. The authors further state the importance of security when considering cloud services that deal with data of potentially competing companies. Concluding their paper, the authors note that various initiatives have been started to enhance security in Industry 4.0. Some initiatives include a proposed security framework for IoT by Cisco[4], guidelines from Fraunhofer[5] or a project by VTT[6].

[4] http://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html
[5] https://www.sit.fraunhofer.de/de/industrie-40/
[6] http://www.vttresearch.com/services/digital-society/cyber-security

Attack Category	Types of Attacks	Loss of	Countermeasures
Gathering	Skimming: quickly reading transmitted messages to collect data	C	Encryption and steganography
	Tampering: deliberately destroying or corrupting data	I	Hash functions, cyclic redundancy checks, and MACs
	Eavesdropping: collecting exchanged messages	C	Encryption, identity-based authentication, and concealed data aggregation (CDA)
	Traffic analysis: monitoring exchanged data to determine traffic patterns	C	Network forensics and misbehavior detection
Imitation	Spoofing: impersonating a user or program to obtain unauthorized access	C, I	Identity-based authentication, key distribution, Internet Protocol Security, and digital signatures
	Cloning: duplicating and rewriting valid data into an equivalent entity	C	Physically unclonable functions
	Replay: recording and storing previously transmitted data to repeat data or delay the current session	C	Time stamps, time synchronization, pseudorandom numbers, session identifiers, and serial numbers
Blocking	Denial of service: Flooding data streams to deplete system resources or interfere with communications	A	Firewalls, router control, resource multiplication, distributed packet filtering, dynamic en-route filtering, and aggregate congestion control
	Jamming: electromagnetic interference or interdiction using the same frequency-band wireless signals	A	Antijamming, active jamming, and Faraday cages
	Malware: Distributing viruses, worms, Trojan horses, spyware, malicious adware, and other programs to interfere with systems	C, A	Antivirus programs, firewalls, and intrusion detection
Privacy	Individual: Deriving a user's locations, preferences, behaviors, and other private information	C	Aggregated proofs, anonymous data transmission, CDA, and advanced digital signatures (e.g., group signatures)
	Group: Deducing an organization's commercial interests and espionage	C	Selective disclosure, data distortion, and data equivocation

Table 2.1: Attack categories, the grouped attacks, possible consequences and potential countermeasures as listed by Ning et al. [NLY13]. The used abbreviations in the Loss of column are: Confidentiality (C), Integrity (I), and Availability (A).

Da et al. in *Internet of Things in Industries: A Survey* [DXHL14] give an overview of the current state of the art in industrial IoT. One section of their work lists enabling technologies for the industrial Internet of Things which includes RFID/NFC and other wireless communication techniques and protocols, smartphones and cloud computing as well as some unexpected technologies such as Barcodes. Also Da et al. highlight the security challenges that arise through the networking aspect of industrial Internet of Things. The authors point out that using certain communication techniques such as NFC offers security-related advantages when compared to other wireless technologies.

2.2.3 Smart Home

In his book *Inside the smart home*, Richard Harper [Har06] gives a definition of smart homes:

> *[...] a home is not smart because how well it is built, nor how effectively it uses space; nor because it is environmentally friendly, using solar power and recycling waste water, for example. A smart home may, and indeed often does, include these things, but what makes it smart is the interactive technologies that it contains.*

As noted by Harper, a smart home is defined by its interactive technologies. He also states that it is easy to include such technology into newly built houses. However, also residents of older buildings might wish to make their homes smart. Thus, wireless technologies to connect devices are proposed in most approaches.

Han and Lim [HL10] suggest to use devices connected using ZigBee [K+03] for efficient energy management. The authors highlight that smart sensors need to be connected to controllers in order to manage the energy consumption of a smart home. One important aspect highlighted by Han and Lim is the security related configuration of devices. They propose to let devices automatically figure out that they need to work together if possible. They further state that some sort of trust center is necessary to only let trusted devices join the network. The authors further discuss that this configuration scenario needs to be possible with as little user interaction as possible.

Konidala et al. [KKYL11] present a security framework for RFID-based applications in a smart home environment. The authors note that most approaches only focus on the pairing and authentication between devices as well as protecting the RFID devices from malicious readers. Konidala et al. conduct a threat analysis regarding RFID devices in a smart home environment and propose a security framework that is capable of mitigating the discussed threats. The authors suggest using electronic product codes (EPC [Bro01]) that are unique identifiers for most security related measures such as authenticating and identifying the RFID tags.

2.2.4 Applications of NFC

Near field communication (NFC) is used in a diverse range of businesses and application domains. Thus, this section gives an overview of some sample use cases that deploy NFC technology. Nowadays, the most well-known application of NFC is in the payment sector, as contactless payment is offered by nearly any payment card provider. Also, the emergence of NFC in current smartphones enables them to be used for mobile payment use cases. Of course, for this domain, security is of utmost importance. Therefore, much research is done regarding the security of mobile, contactless payments.

For instance, Pasquet et al. [PRR08] introduce a prototype for mobile payment using smartphones which complies with European laws. For the implemented prototype, a security analysis is given by the authors. The authors conclude that providing security on all levels (physical, software, and protocol) is nearly impossible for a single institution. Therefore they state, that a collaboration of multiple partners for security relevant (NFC) projects should be desired.

Another very prominent field for NFC-based solutions are smart homes and the Internet of Things (IoT). Mohsen and Michael [DM08] state that dummy things can be made smart things by equipping them, for instance, with RFID. The authors propose a so-called *Smart Home Master-Slave RFID System Architecture* which comprises several RFID readers in a master-slave architecture. The system consists of a master reader, which is used as a gateway to the Internet, a couple of slave readers which are used to extend the reading range of the master reader and mobile readers which can be operated at varying positions in the smart building. The authors note that by using that architecture, a smart home with up to 100% reader coverage can be build. Using this readers, several smart services such as shopping, monitoring or healthcare applications become possible.

A third very popular area of operations for NFC is in ticketing [NFC11a]. Ghìron et al. [GSMM09] present *NFCTicketing*, an NFC-based ticketing system for public transportation. In their prototype, the authors use an NFC-enabled mobile device that also is equipped with a secure element which is used to store ticket information. The phone's NFC module is used for purchasing tickets as well as for validating the ticket (by the ticket inspector). The authors included a user study regarding the *satisfaction* of users concerning the NFC ticketing prototype, with more than two-thirds of the participants stating either excellent or good satisfaction with the proposed technology.

Steffen et al. [SPS+10] discuss use cases, architectures, and realizations of NFC in an automotive environment. According to the authors, all described use cases have been implemented (as prototypes) in a BMW vehicle. They list different use case categories:

- **Communication Support**: e.g., Bluetooth and WiFi pairing or electronic business cards (vCards)
- **Interaction between the Customer and the Car**: e.g., personalization (sound settings, seat and mirror settings, ...) as well as user authentication and car access
- **Information Retrieval**: e.g., car informations and spare part informations
- **Car Key with NFC Interface**: e.g., car status, e-ticketing using car key

The presented architecture gives an overview of how the necessary NFC hardware can be integrated into existing bus systems of current (BMW) vehicles. All use cases mentioned in their work are also implemented in a prototype based on a series-production BMW.

Lahtela et al. [LHJ08] discuss the applicability of RFID and NFC for the healthcare sector. By using information technology in healthcare, the safety of patients should be increased according to the authors. A system based on NFC for the Kuopion University Hospital is proposed that should decrease medication errors by connecting sensors to an automatic medication dispenser system. By identifying medication and patients with RFID tags, pharmacists' errors could be further minimized and thus, patients' safety increased as concluded by the authors.

Other related work includes using an NFC-enabled smartphone for access control, as proposed by Dimitrienko et al. [DSTW12]. The authors use smartphones for access control instead of the more common approach of using smartcards. Aigner et al. [ADF07] present an NFC-based system that enables secure virtual coupons. By using virtual coupons, the problem of customers illicitly copying coupons is mitigated by using passive tags in coupons. Finally, lecture attendance of students at the Budapest University of Technology and Economics is monitored using an NFC-based approach [BSDF12]. NFC enabled student cards in combination with biometric information were used to track student attendance and to autonomously allow or prohibit students from successfully finishing courses which required a certain amount of attendance.

2.2.5 Bring Your Own Device

The concept of *Bring Your Own Device (BYOD)* more or less forced into the minds of employers by their own employees, as Gordon Thomson [Tho12] states:

> *Ten years ago, employees were assigned laptops and told not to lose them. They were given logins to the company network, and told not to tell anyone their password. End of security training. Today, your 'millennial' employees [...] show up to their first day on the job toting their own phones, tablets, and laptops, and expect to integrate them into their work life.*

However, this BYOD movement also imposes new security risks for IT networks as the devices are to a large degree inhomogeneous and therefore managing them, for instance, via security policies, becomes a hard task. As stated by Miller et al. [MVH12], security risks do not only include the infection of company networks by malicious applications coming from employee's devices. A second factor is that by integrating personal devices into the network, sensitive data might make its way onto devices from where it can be synced to arbitrary places on the Internet.

Many publications focus on improving security in the BYOD context. For instance, a work done by Armando et al. [ACM13] proposes a security framework for mobile devices that tries to ensure that only applications in line with companies' security guidelines can be installed on a mobile device. Instead of using, for instance, static blacklists, the behavior of applications is analyzed and validated against security policies. A prototypical implementation of their proposed framework is presented which is applicable to Android phones.

2.2.6 Bring Your Own Key

Based on the term *Bring Your Own Device*, a different concept named *Bring Your Own Key* was introduced as noted by Hongwen Zhang [Zha15]. The first use cases for BOYK were in cloud computing, as motivated by the authors:

> *[...] if we are encrypting data because we are worried about its security in an unknown cloud, why then should we trust the same cloud to hold the encryption keys?*

In addition to the key, a more sophisticated approach also suggests to *Bring Your Own Encryption*, which is an extension to BOYK. Instead of only using a customer's own key for cloud services, also own encryption algorithms can be applied. As Syed and Ussenaiah [SU15] note, by doing so, only the ciphertext needs to be stored on cloud service providers' physical storage. Thus, customer's have control over their keys and their own master key, as long as a *Hardware Security Module (HSM)* is used.

2.2.7 Configuration via QR Codes

As mentioned earlier, QR codes can store arbitrary data, thus making them suitable for a broad range of use cases where data can be transferred from the QR code to a mobile device. One such application (which proposes NFC as an alternative) allows customers to configure individual tickets for public transport [FT11]. In the system proposed by Finžgar and Trebar, QR codes are used to determine the starting and end point of a ticket. Based on those locations, the fare is then calculated and billed to the customers.

In another work, Lee and Han [LH12] state that although a very accurate approach for indoor localization using Wi-Fi, fingerprinting requires an enormous amount of effort for calibrating. QR codes are attached at fixed positions within the service area, thus enabling every user in possession of a smartphone to contribute to the collection of Wi-Fi fingerprints. When scanning a respective QR code, all Wi-Fi signals, and the corresponding signal strength as well as the position encoded within the QR code are sent to a server and used to update the radio map used for Wi-Fi based indoor localization.

Starnberger et al. [SFG09] show the application of QR codes to a system where security is of utmost importance: mobile transaction authentication. The authors propose to use QR-TANs which according to them allows users of untrusted networks and terminals to validate and approve transactions. Security is enabled by a challenge-response mechanism where the challenge and a nonce is transmitted to the user's mobile device by displaying a QR code. The response is then displayed on the mobile device's screen and is only a few letter code which users need to type into the terminal to approve the transaction.

2.2.8 Configuration via NFC

When considering configuration via NFC, a distinction between two different configuration processes needs to be made. The first case, where configuration parameters are transferred from a mobile device to a second device via NFC is the approach that is similar to the one applied in this master's thesis. The second method is to store configuration parameters in an RFID tag from which configuration parameters can be read by (and applied to) a mobile device.

The first approach where (interchangeable/dynamic) configuration parameters are transferred from a mobile device to another device is not yet widely discussed in research. However, a work by Haase et al. [HMEK16] introduces wireless sensor and actuator configuration via NFC. The authors not only present a theoretical approach on how to use NFC to configure sensors and actuators but also show a prototypical implementation of their concept. The prototype comprises an NFC-enabled house chip used in-house automation and an Android application for the mobile device.

The second type of NFC-enabled configuration is far more widespread than the one discussed first. Using RFID tags to transfer configuration parameters containing Bluetooth or WiFi pairing information to a mobile device is a concept used quite often nowadays. For instance, Matos et al. [MRT12] propose a secure WiFi configuration method using NFC. In their prototype, the authors demonstrate how to provide an secure NFC side channel using public key cryptography. Using this side channel, connection information like credentials and the necessary network ID are transferred.

In another approach, López-de-Ipiña et al. [LdIVJ07] use NFC in their *Touch2Launch* service to configure a Bluetooth connection between a mobile client and a so-called *Sentient Graffiti* server. By placing the RFID tags, virtual graffiti (arbitrary messages) can be placed and read from the associated RFID tags. The corresponding data is stored in a central unit which can then be queried using the established Bluetooth connection.

A hybrid approach that combines both mentioned configuration types is presented by Christensen and Wagner [CW08]. In their work, an already existing healthcare framework (*Healthcare@Home*) that includes the automatic pairing of wireless sensors with a base station is extended to support NFC-based configuration. In the discussed use case, the framework allows inexperienced users to set up a wireless health monitoring system. In the proposed system, a smartphone is used to pair the wireless sensors with a base station. To do so, information such as MAC address and public key are read from the wireless sensor using NFC. This information is then transferred to the base station via the NFC-enabled smartphone. Using this information, a WiFi or Bluetooth connection between the sensor and base station can be established.

2.3 Differences to the State of the Art

In this section, the differences of the solution designed and implemented in this master's thesis are compared to other, state of the art solutions that were already mentioned in related work in Section 2.2. The improvements of the presented approach compared to those other technologies will be highlighted.

2.3.1 Approaches using QR codes for configuration

The approaches presented in [FT11], [LH12] and [SFG09] all use QR codes to transfer small amounts of data such as geolocations or TANs. All solutions except one do not consider security at all. In contrast to that, the concept designed and implemented in this master's thesis uses QR codes to transfer arbitrary data of a size up to the maximum allowed for QR codes (see Section 2.1.4 for details). The main improvement, however, when comparing to those previously mentioned approaches is the consideration of security aspects as also confidential data might be stored in a QR code. By encrypting the data, arbitrary, confidential information can be transferred using QR codes.

2.3.2 Approaches using NFC for configuration

In Section 2.2.8 a wide range of concepts that use NFC for configuration purposes was presented, such as [MRT12], [LdIVJ07] and [CW08]. Those concepts use NFC to transfer configuration parameters, mostly between two devices using some sort of

mobile device for the intermediate transportation. This concept is similar to the one presented in this thesis. However, the configurations transferred in those approaches are mostly parameters for device pairings of wireless connections, such as Bluetooth or WiFi. In contrast to those solutions, the one designed and implemented in this master's thesis allows transferring arbitrary configuration parameters, such as production relevant settings.

One approach presented by Haase et al. [HMEK16] suggests to use mobile phones to configure wireless sensors and actuators. Although the concept is very similar to the one presented in this thesis, the work only mentions possible security considerations without going into detail. In contrast to that, the work done in this master's thesis presents an approach where arbitrary configuration data can be transferred using mobile devices in a secure way. It specifies a methodology on how to secure confidential data while being transferred and also suggests the inclusion of hardware security elements to further strengthen the protection of the transported data.

2.3.3 Combination of Approaches

All previously discussed methods either transport configuration data using QR codes or NFC technology. However, for the transport of information between two devices, the QR code approach is not practicable as at least one of the two devices would need to be equipped with a camera. Using NFC might be suitable in a business customer scenario, but requiring users in a smart home scenario to purchase NFC readers to be able to configure their smart sensors using NFC might not be desired. Therefore, the concept presented in this master's thesis suggests combining those two technologies. By using QR codes to transfer data to a mobile device which then uses NFC to transmit the data to the device which should be configured, those drawbacks will be mitigated.

2.3.4 Overview

In Table 2.2 an overview of the previously discussed differences to state of the art related work is given. Certain aspects that are required for an NFC-based configuration interface for smart sensors are highlighted. In comparison, the approach presented in this master's thesis fulfils all listed requirements.

Related Work	Arbitrary Payload	Payload Secured	Practical for Smart Factories	Practical for Smart Homes	Additional HW necessary
[FT11], [LH12], [SFG09]	-	-	-	-	cameras at devices to be configured
[MRT12], [LdIVJ07], [CW08]	-	O	O	O	NFC reader needed to transfer payload to mobile device
[HMEK16]	+	O	+	O	NFC reader needed to transfer payload to mobile device
Master's Thesis	+	+	+	+	none

Table 2.2: Overview of related work which highlights certain requirements that are necessary for a NFC-based configuration interface.

3 Design

In this chapter, possible application scenarios of NFC-based configuration are presented. After that, the core system architecture of the NFC-configuration interface developed for this master's thesis will be discussed. Based on the general design, the communication protocol to transfer the configurations will be introduced. Concluding this chapter, the design of all involved system components is going to be discussed in detail. To address the design of each system component, standard UML artifacts such as class diagrams, component diagrams, and sequence diagrams are used [RJB04].

3.1 Application Scenarios

In this section, three exemplary application scenarios for the NFC-based configuration interface presented in this master's thesis are discussed. All three application scenarios fit within the scope of the *IoSense* project.

3.1.1 Bring Your Own Key scenario

As already introduced in Section 1.2, a hardware component called *Mediator* was presented by Lesjak et al. [LRB+14b]. This device can be used to allow the application of smart maintenance services for (legacy) manufacturing devices. The Mediator hardware is connected to any existing machine using its equipment interface as shown in Figure 1.3. Device information such as operating hours or other maintenance relevant information (called *snapshot* in that context) are then fetched by the Mediator. This information is then transmitted using a secured version of the MQTT protocol [LHH+15]. Because

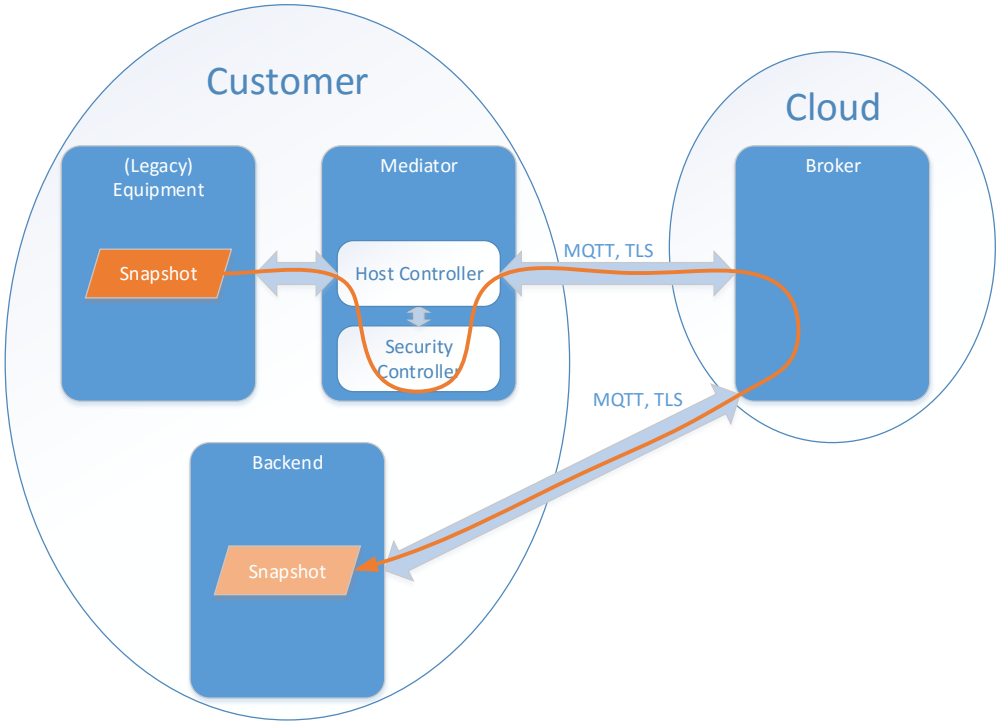


Figure 3.1: Bring Your Own Key scenario. The transferred data needs to be end-to-end encrypted to not be read by cloud providers or other customers.

MQTT is a publish-subscribe protocol, the information needs to be transferred to a broker which handles the aforementioned publish-subscribe functionalities. As discussed by Lesjak et al. [LHH+15], by securing the MQTT protocol with TLS, subscribers to the topic can be authenticated using certificates. In this scenario, a smart maintenance provider might have access to maintenance relevant data from multiple customers.

When equipping manufacturing devices with networking capabilities, also production relevant information could be transferred using the same technology. However, some information might be confidential and not intended to be read by the smart maintenance provider. Furthermore, as the data of multiple customers can be processed at a single broker, also possible competitors' data could be stored at the same broker. Therefore, customers might be reluctant to applying this technology because they can not be sure that their confidential data can not be read by others as the smart maintenance provider might share the same key for multiple customers and thus, any customer could subscribe to a competitor's data and decrypt the provided information.

In this context, a concept named *Bring Your Own Key* that already was discussed in Section 2.2.6 can be applied. If a customer provides and deploys its own keys for encryption and decryption, no other party will be able to read the transferred information. The sce-

nario showing this end-to-end encryption is depicted in Figure 3.1. There, confidential information from a customer's manufacturing equipment is transferred to an MQTT broker where no information about other subscribers is known. By providing the keys necessary for encryption and decryption and by keeping those keys secret, a customer is guaranteed that no other party will be able to read the information. In such a scenario, the configuration of keys can be done using a NFC-based configuration interface. Thus, allowing customers to deploy keys by simply touching both devices which should be configured to have the same private key. By doing so, no other party needs to know or process any key, therefore guaranteeing that the key is only known by the party deploying it to the involved devices.

3.1.2 Industry 4.0

The second application scenario, smart factories in the context of Industry 4.0, are described by Shrouf et al. [SOM14]:

> *New market requirements and emerging autonomously technologies such as IoT are shifting the manufacturing companies' environment toward smart factories. The basic idea of IoT is a system where the physical items are enriched with embedded electronics (RFId tags, sensors, etc.) and connected to the Internet.*

As noted by the authors, physical items can be equipped with embedded electronics, which offers a manifold range of possibilities in the Industry 4.0 context. Not only can the manufacturing devices be connected to each other and the Internet, but also to the wrought products that are currently produced. Humans that are monitoring, controlling and maintaining the system are then able to perform all of those activities using wireless devices such that having direct physical access will not be necessary.

In the context of the IoSense project, a smart sensor should be configured during its whole life cycle, starting during the manufacturing process. By using an NFC-based configuration interface, the configuration of smart sensors can be done wireless while the sensor is assembled, for instance, on a production line. This scenario is depicted in Figure 3.2 where also the Industry 4.0 vision is illustrated. There, the manufacturing machines communicate with each other while being monitored, controlled and even maintained by human workers utilizing the wireless interface. The machines also apply configurations to the produced items during manufacturing, enabling the products to be highly customized as envisioned in an Industry 4.0 scenario.

However, the usage of this NFC-based configuration interface, of course, is not restricted to configuration during the manufacturing process. As previously mentioned, one goal of the IoSense project is to enable smart sensors to be configurable during their whole life cycle. By providing technologies to utilize the included NFC interface for applying

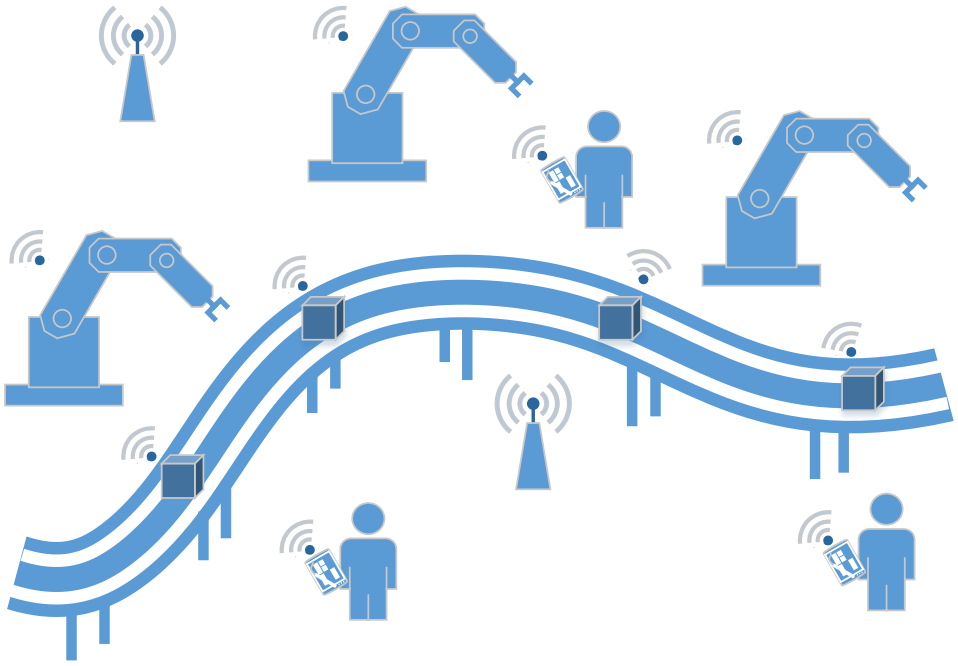


Figure 3.2: Smart factory scenario in Industry 4.0. Machines communicate with each other and are configured by maintenance workers. The produced sensors are configured using NFC during the manufacturing process.

configurations, this requirement can be fulfilled.

Another more recent trend in Industry 4.0 research is the integration of autonomous robots into smart factories. Many research institutions, as well as companies, are doing research in that context, as Rüßmann et al. [RLG+15] note:

> *Kuka, a European manufacturer of robotic equipment, offers autonomous robots that interact with one another. These robots are interconnected so that they can work together and automatically adjust their actions to fit the next unfinished product in line.*

Plosz et al. [PFT+14] analyzed various wireless technologies concerning industrial usage. In their analysis, NFC was suggested as being a good choice in that context because of its short communication range. Therefore, in the scenario depicted in Figure 3.3, autonomous robots are shown that transport wrought material between manufacturing machines. As highly customized products will be produced in such a setting, robots need to configure the manufacturing machines to account for the currently delivered wrought product.

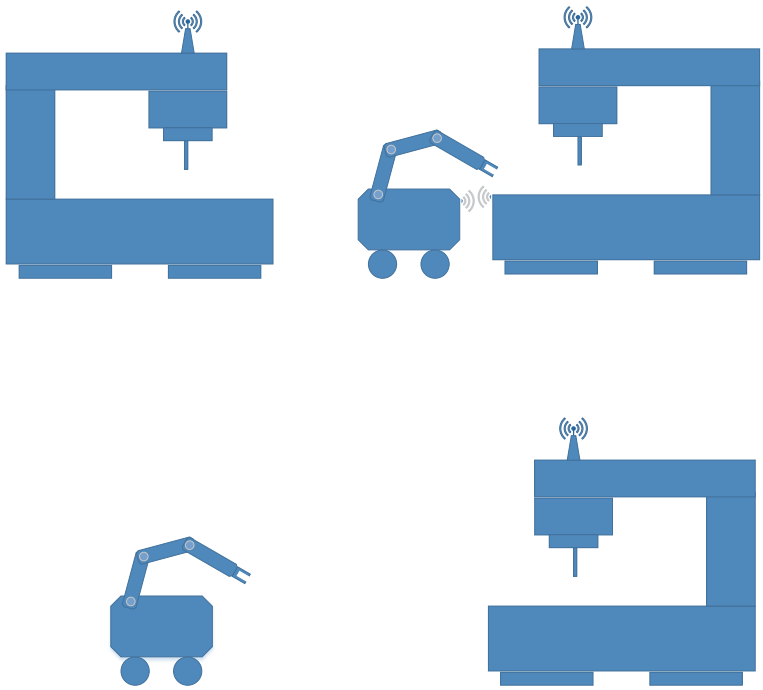


Figure 3.3: Smart factory scenario in industry 4.0. Autonomous robots transport wrought material between manufacturing machines.

Local configuration instead of a central instance that configures all machines needs to be performed as the travel time of autonomous robots is not deterministic due to unforeseen events such as an interfering human. Due to its short communication range and due to that useful security features, as well as to prevent overlapping and hence interfering wireless channels, an NFC-based configuration interface can also be used in this application scenario.

3.1.3 Smart Home

The third presented application scenario is in the context of smart homes where various devices that are used in everyday's life need to be configured. Cook et al. [CYHI+03] present a smart home vision for their project named *MavHome*:

> *At 6:45am, MavHome turns up the heat because it has learned that the home needs 15 minutes to warm to optimal waking temperature. The alarm sounds at 7:00, after which the bedroom light and kitchen coffee maker turn on. Bob steps into the bathroom and turns on the light. MavHome records this interaction, displays the morning news on the bathroom video screen, and turns on the shower. When Bob finishes grooming, the bathroom light turns off while the kitchen light and display turn on, and the news program moves*

to the kitchen screen. During breakfast, Bob requests the janitor robot to clean the house. When Bob leaves for work, MavHome secures the home, and starts the lawn sprinklers despite knowing the 30% predicted chance of rain. Because the refrigerator is low on milk and cheese, MavHome places a grocery order. When Bob arrives home, his grocery order has arrived and the hot tub is waiting for him.

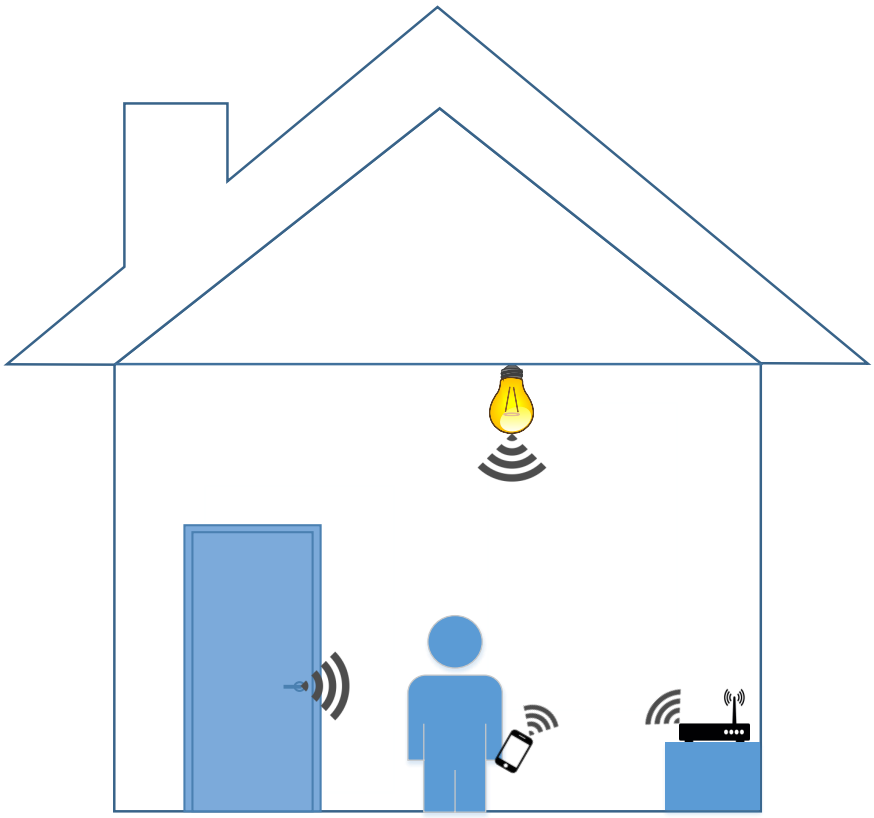


Figure 3.4: Smart home scenario. Various sensors and actuators can be configured and controlled wirelessly by the user.

Based on that scenario, devices such as the heater, alarm, coffee maker and so on need to be configured. For the same reasons as in an industrial context (security due to short communication range and low power consumption), an NFC-based configuration interface can be used for this configuration tasks.

3.2 System Architecture

To implement an NFC-based configuration interface, the following three components are necessary, for which the general system architecture will be discussed in this section.

Device: The device that gets configured and therefore needs to be equipped with a secure element. By utilizing the included NFC interface, configuration data can be transferred wireless, while by using the equipment interface (see Figure 1.3) any (legacy) device can be connected with the secure element.

Backend: The backend is used to prepare and manage configuration data. Newly created configurations are encrypted and provided for transportation by the backend. Therefore, also the key material of all managed devices needs to be maintained.

Mobile Devie: The mobile device is used to transport configuration data provided by the backend to a receiving device.

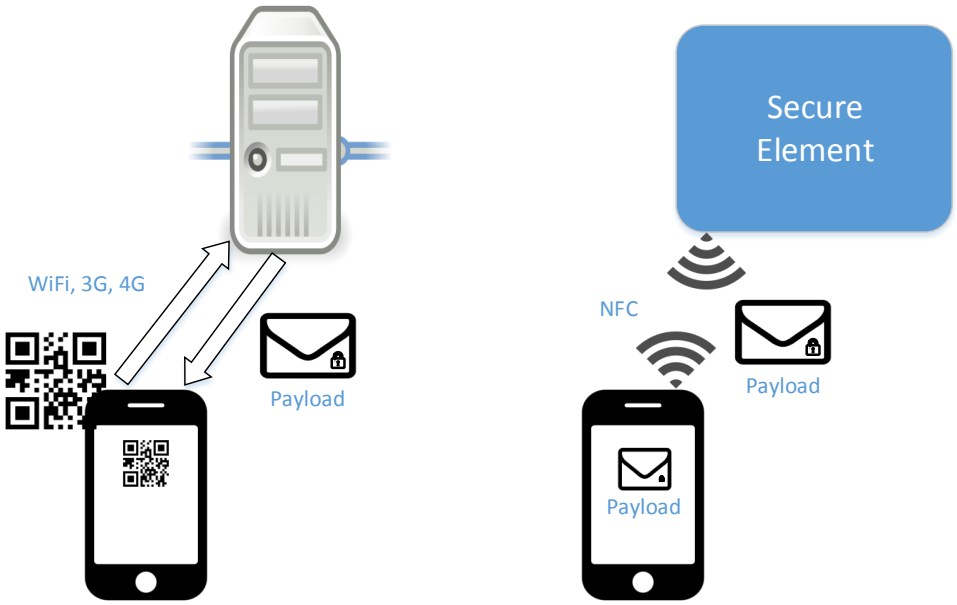


Figure 3.5: System architecture of NFC-based configuration scenario that is suitable for industrial as well as for smart home use-cases. QR codes are used to transfer configurations to a mobile device, while NFC is used to transport configurations to the recipient device.

When deciding on which interfaces to use for data transport, the industrial as well as the end-user smart home use-cases need to be considered. In an industrial setting, using NFC to transfer data from a central device that is managing the configurations to a mobile device and from there using NFC to the device that is being configured might be suitable. However, when also considering the use-cases related to end-users, a dedicated

NFC-device would be needed in addition to the mobile device which is used to transport the configurations. Therefore, the system architecture depicted in Figure 3.5 was chosen.

As can be seen in Figure 3.5, Quick Response (QR) codes are used to transfer configurations to the mobile device, while NFC is used to transport the configurations stored on the mobile device to the recipient device. The design of both implemented protocols will be discussed separately in Section 3.3.

3.3 Protocols

The system architecture depicted in Figure 3.5 illustrates the two different transportation mechanisms used. Because the techniques used to transport the configuration differs regarding the respective data transfer direction, separate protocols for QR-based and NFC-based are introduced in this section.

3.3.1 QR-based

The QR-based transport protocol is used to transfer configuration parameters from a backend to the mobile device. The proposed protocol is based on JSON and has the following mandatory fields:

```
1 {
2   "type" : "inline|url",
3   "title" : "configuration title",
4   "payload" : "..."
5 }
```

Listing 3.1: JSON package used inside QR-codes.

The `title` field must contain a descriptive title associated with the configuration package. Although the title is not transferred when configurations are transported, it is essential to display this information on the mobile device to be able to distinguish between multiple configurations. The field `type` defines if the configuration package is contained in the QR codes payload (`inline`) or if the data needs to be fetched separately from the backend (`url`).

The reason to allow those two different modes is that if the configuration including the overhead imposed by this JSON file does not exceed the maximum capacity of a QR code, no additional connection is needed. This would allow the *offline* configuration of devices by just equipping a maintenance worker with a set of printed QR codes. If, however, the payload is too large to fit in a QR code or an explicit connection to fetch the

configuration parameters is desired, the second type can be chosen.

Depending on the defined type, the content in the `payload` field will differ. If the type is defined as being `inline`, the payload will directly contain the payload which can be transferred to the receiving device. In the case of `type=url`, the payload contains a URL from which the payload needs to be fetched first. Also, the security measures applied differ between those two types. The differences will be discussed in Section 3.5.

3.3.2 NFC-based

Configurations are transmitted via NFC using NDEF messages. For devices to support NDEF, the Type 4 Tag Operations need to be implemented. All operations that are necessary and their according structure are discussed in Section 3.4. The structure of the NDEF message embedded in those Type 4 Tag Operations can be seen in Table 3.1.

Mobile Device Realtime	**Cipher Specs**	**Encrypted Payload**	**MAC**
4B	2B	(NDEF length - MAC length- 6) B	according to *Ciper Spec* B

Table 3.1: Structure of the NDEF message that is sent when configuring a device via NFC.

- **Mobile Device Realtime**: The real time of the mobile device in milliseconds when the NDEF package is transmitted. This field is used for security measures and will be explained in detail in Section 3.5.
- **Cipher Specs**: The used cryptographic algorithms and corresponding key lengths when encrypting the payload are specified here, such that the decryption and verification can be done with the correct specifications.
- **Encrypted Payload**: The payload encrypted using the cryptographic algorithm specified in the Cipher Specs field. The length of the encrypted payload depends on the chosen algorithms and key lengths for encryption and MAC.
- **MAC**: The message authentication code (MAC) calculated using the cryptographic algorithm specified in the Cipher Specs field. The length of the MAC depends on the chosen algorithm and key length.

3.4 NFC Type 4 Tag Operations

To transmit data between the mobile device and the secure element, NFC Type 4 Tag Operations need to be supported by the secure element in order to communicate at a higher abstraction level. According to the Type 4 Tag Operation Specification [NFC11b], an NFC Forum Device *shall* support the commands:

- **Select**: Selection of applications or files to read/write from/to.
- **ReadBinary**: Read data from a selected application or file.
- **UpdateBinary**: Write data to a selected application or file.

These three commands are sent via command APDUs (C-APDU). APDUs (application data protocol unit) are the transferred packets in the context of smart cards. Replies to commands are sent via so-called response APDUs (R-APDU). If all three commands are implemented correctly on a tag, NDEF messages (NFC data exchange format) can be transferred to and from the tag. A C-APDU is defined as depicted in Table 3.2 while the structure of a R-APDU is shown in Table 3.3. The general procedure is to always first select a file and then either read from or write to that previously selected file.

CLA	INS	P1	P2	Lc (optional)	Data (optional)	Le (optional)
1B	1B	1B	1B	0B, 1B, 3B	Nc B	0B, 1B, 2B, 3B

Table 3.2: Structure of a C-APDU.

- **CLA**: Instruction class which indicates the class of a command.
- **INS**: Instruction code which refers to a specific command.
- **P1-P2**: Instruction parameters 1 and 2.
- **Lc**: Contains the number of bytes (Nc) of data. If 1 byte is used, Nc is in the range [1, 255], if 3 bytes are used, the first byte must be zero, therefore Nc is in the range [1, 65535].
- **Data**: Nc bytes of data.
- **Le**: Maximum number of response bytes (Ne) expected. If 0 bytes used, Ne=0. 1 byte denotes a range of 1 to 256 (0 not included). 2 (if Lc was present) or 3 bytes (if Lc not present) denote the same range as in the 3 byte Lc case.

- **Response body**: A maximum of Ne bytes response data.

Response Body (optional)	SW1	SW2
$\leq$ Ne B	1B	1B

Table 3.3: Structure of a R-APDU.

- **SW1-SW2**: Command response code, e.g., 0x9000 is indicating success.

To detect a potential NDEF data transfer, the tag must provide a so-called capability container (CC) which can be read when sending a *ReadBinary* C-APDU for file *E103h*. The most important fields contained in the capability container include:

- **MLe**: Specifies the maximum size (in bytes) that can be read from the type 4 tag.
- **MLc**: Defines the maximum size (in bytes) that can be sent to the type 4 tag.

To read the capability container, the corresponding file needs to be selected first, followed by a *ReadBinary* C-APDU. The corresponding *SelectFile* C-APDU is depicted in Table 3.4. To read from a previously selected file, a *ReadBinary* command needs to be sent. An example of such a command is shown in Table 3.5. An example C-APDU to write to the type 4 tag (*UpdateBinary*, after previously selecting a file with the *Select* command) is shown in Table 3.6.

CLA	INS	P1	P2	Lc (optional)	Data (optional)	Le (optional)
0x00	0xA4	0x00	0x0C	0x02	0xE103	-

Table 3.4: Example C-APDU to select capability container.

CLA	INS	P1	P2	Lc (optional)	Data (optional)	Le (optional)
0x00	0xB0	Offset		-	-	Length Ne

Table 3.5: Example C-APDU to read data from previously selected file.

CLA	INS	P1	P2	Lc (optional)	Data (optional)	Le (optional)
0x00	0xD6	Offset		Datalength Nc	Data	-

Table 3.6: Example C-APDU to write data to previously selected file.

3.5 Configuration Security Mechanisms

In this Section, all implemented security measures are discussed. Those measures include using authenticated encryption, configuration version management, the limited validity of configurations, and machine identifiers.

3.5.1 Order of Encryption

As discussed in Section 2.1.5, there are three different modes to implement authenticated encryption: *Encrypt-then-MAC*, *Encrypt-and-MAC*, and *MAC-then-Encrypt*. Hugo Krawczyk [Kra01] analyses those three methods concerning security when combining arbitrary symmetric ciphers with secure hash functions. He states that although both components may be perfectly secure, the combination of them makes the authenticated encryption vulnerable to attacks. However, the author points out that *Encrypt-then-MAC* is not vulnerable to attacks when combining certain cryptographic algorithms with specific (secure) hashes. Two main principles which should be followed to make authenticated encryption *strongly unforgeable* [BN00]:

1. Never hash any plaintext, always apply hash functions to ciphertexts.
2. Never use the same key for encryption and hashing.

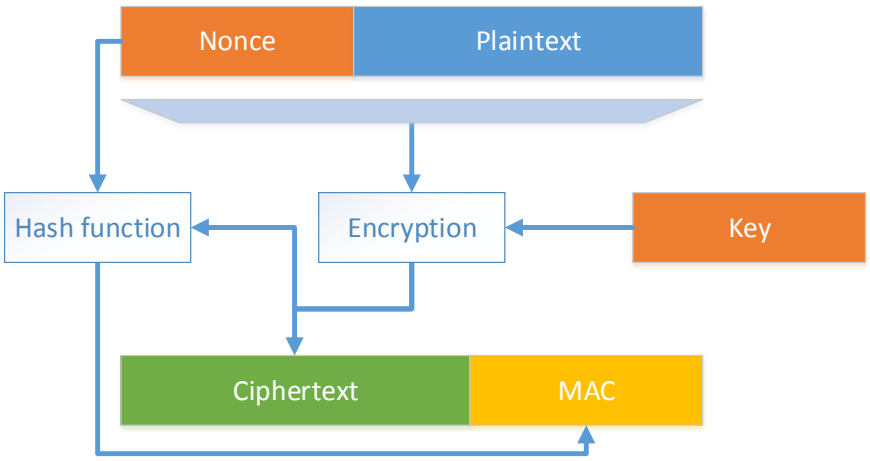


Figure 3.6: Principle of authenticated encryption scheme that implements the Encrypt-then-MAC method using nonces to generate the MACs.

The first principle is applied by using Encrypt-then-MAC. The second principle will be satisfied by generating a *nonce* which is used when calculating the MAC. A (cryptographic) nonce is any arbitrary number which will only be used once. The applied principle is depicted in Figure 3.6. As can be seen there, the nonce and plaintext are con-

catenated and encrypted. The resulting ciphertext and the previously generated nonce are then used to calculate the MAC, which is attached to the ciphertext.

3.5.2 Version, Validity, and Machine ID

The encrypted payload which is included in the NDEF protocol discussed in Section 3.3.2 actually consists of multiple parts which are shown in Table 3.7. After decrypting the encrypted payload, the different parameters are used to decide if the configuration will be accepted.

HMAC Nonce	Version	Validity	Machine ID	Plaintext
depending on Cipher Specs B	2 B	4 B	4 B	(Payload Length - 10 - Nonce Length) B

Table 3.7: Structure of the content contained in the encrypted payload.

- **HMAC Nonce**: Used to calculate a MAC of the received encrypted payload which is then compared to the MAC transmitted in the NDEF message. The nonce is generated at the backend when the MAC is calculated. If the two MACs do not match, the configuration is discarded.
- **Version**: The version number must be higher than the current version number. Otherwise the configuration will be discarded. The backend needs to keep track of the version number of each managed device.
- **Validity**: The time until which this configuration must be transferred from the mobile device to the machine which needs to be configured. The validity is based on the mandatory real time of a mobile device which is transmitted when a configuration is fetched from the backend. This means that no validity can be specified for *inline* QR codes (see Section 3.3.1). If the *Mobile Device Realtime* which is transmitted in the NDEF message (see Section 3.3.2) is larger than the validity time-stamp, the configuration is discarded.
- **Machine ID**: If the transmitted machine ID is different from the actual machine ID, the configuration will be discarded. The machine ID is specified by the backend when generating the corresponding configuration.
- **Plaintext**: The actual configuration parameters as key-value pairs.

3.6 Backend Design

The backend application is implemented as a web application using JavaScript and the libraries listed in Section 4.1.2. Therefore, the class diagram depicted in Figure 3.7 shows a class diagram of all implemented classes and the used interfaces provided by those libraries. The following list gives an overview of each class' responsibilities:

Main: The web application's main class which is the entry point of the application when it is launched. This class also includes the user interface components.

ConfigEntry: One entry of a configuration. It is represented as a key-value pair which is capable of handling any data that can be serialized to a byte stream.

Config: The class representing a configuration. It consists of at least one configuration entry. No duplicate keys are allowed in a configuration.

ConfigStorage: All configurations, the corresponding version IDs as well as the encryption keys are stored in this class. The confidential data is kept in a secure storage by this class.

QRCodeGenerator: This class generates QR codes based on the given information and the protocol presented in Section 3.3.1.

CryptoJS.AES: This class handles all AES functionalities such as encryption and decryption of given inputs. When encrypting, an initialization vector, as well as a block cipher mode, need to be specified.

CryptoJS.HMACSHA256: This class provides the MAC functionality based on the HMAC-SHA256 algorithm. When calculating a hash, the input text, as well as a nonce, are needed.

pem: Because self-signed certificates are used, the pem module generates a certificate when the application is started.

express: This module handles all requests such as GET or POST made to the application. For each desired request, the method and the corresponding path, as well as a callback function, need to be specified.

https: The HTTPS module creates the socket needed for a web application. Because only secure HTTPS traffic is allowed, a certificate is required in addition to the port when creating the socket.

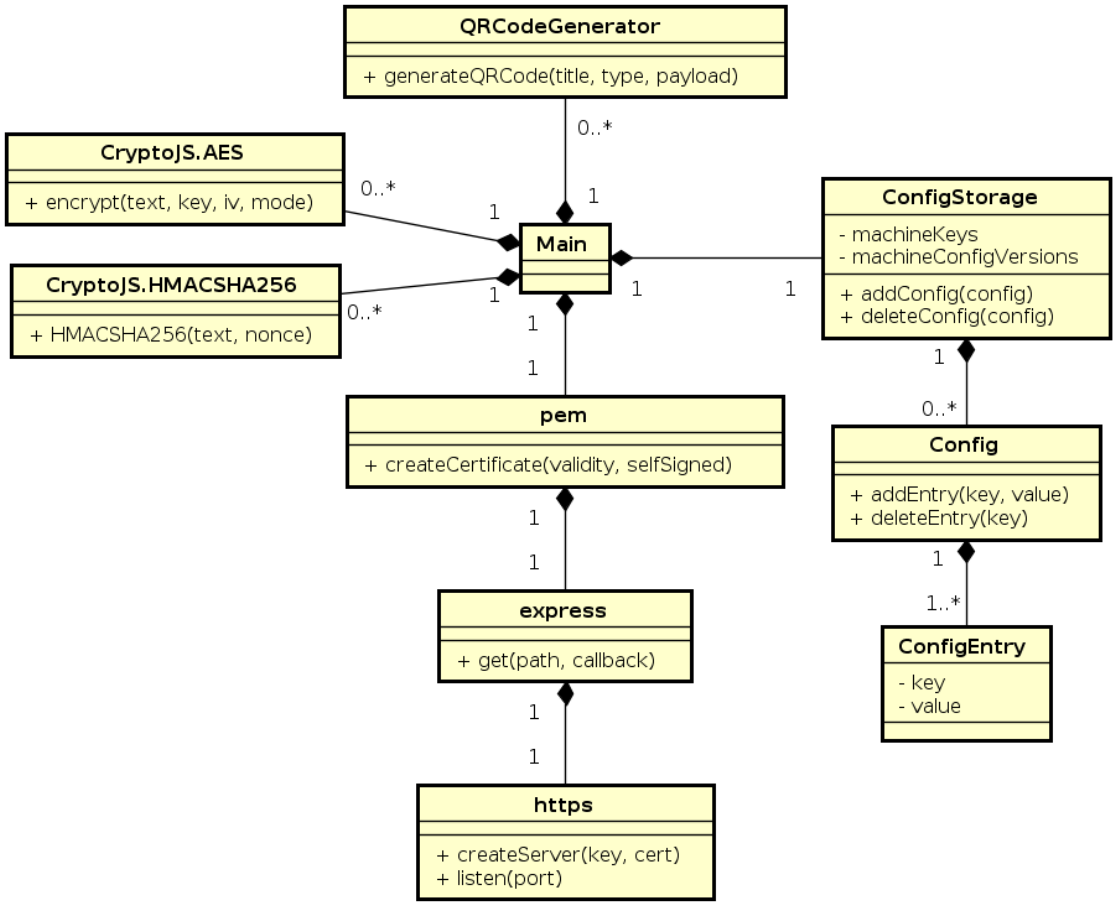


Figure 3.7: Class diagram of the backend web application.

3.7 Mobile Device Design

The design of the application developed for the mobile device, SECURECONFIG, will be discussed using a class diagram as well as sequence diagrams that highlight the invocation of certain components.

3.7.1 Class Diagram

As the SECURECONFIG application was implemented for Android OS, operating system specific requirements had to be considered during the design phase. The corresponding class diagram is depicted in Figure 3.8. The tasks of each class will be explained in the following list:

MainActivity: Each Android application needs a main entry point (*onCreate*) in a main class. For the SECURECONFIG application, the MainActivity class is this entry point. The classes responsibilities are mainly the handling of user interface related events such as button clicks or the pausing and resuming of the application.

ConfigurationListAdapter: To display classes different than Strings in a list, a custom list adapter class needs to be integrated. The responsibility of this class is to keep track of the underlying data source and to notify the user interface thread of changing entries.

IntentIntegrator: To invoke the third party QR code scanner application, a custom intent needs to be implemented which is then passed to the Android operating system. The scanner application registers for this intent and launches on detection of such an intent.

IntentResult: If a QR code was scanned successfully by the third party application, the corresponding text is returned to the calling application as an IntentResult.

Configuration: The class representing the actually transferred configuration. Here, the encrypted configuration data, as well as metadata such as a title are stored.

ConfigurationStorage: A list of currently stored configurations. This class is responsible for storing all available configurations.

QRCodeParser: This class is used to parse the text that is received when reading a QR code. The corresponding protocol was already presented in Section 3.3.1.

DownloadConfigAsyncTask: If a configuration needs to be fetched from the backend, this class is used. The class implements the interface of an asynchronous task and therefore can be invoked in the background. The reason for this is that Android does not allow to start downloads in inside the main thread.

NfcReaderWriter: This class handles the NFC communication with the secure element. This includes registering a callback function for the detection of new tags as well as providing read and write operation for such tags. The corresponding protocol was already presented in Section 3.3.2.

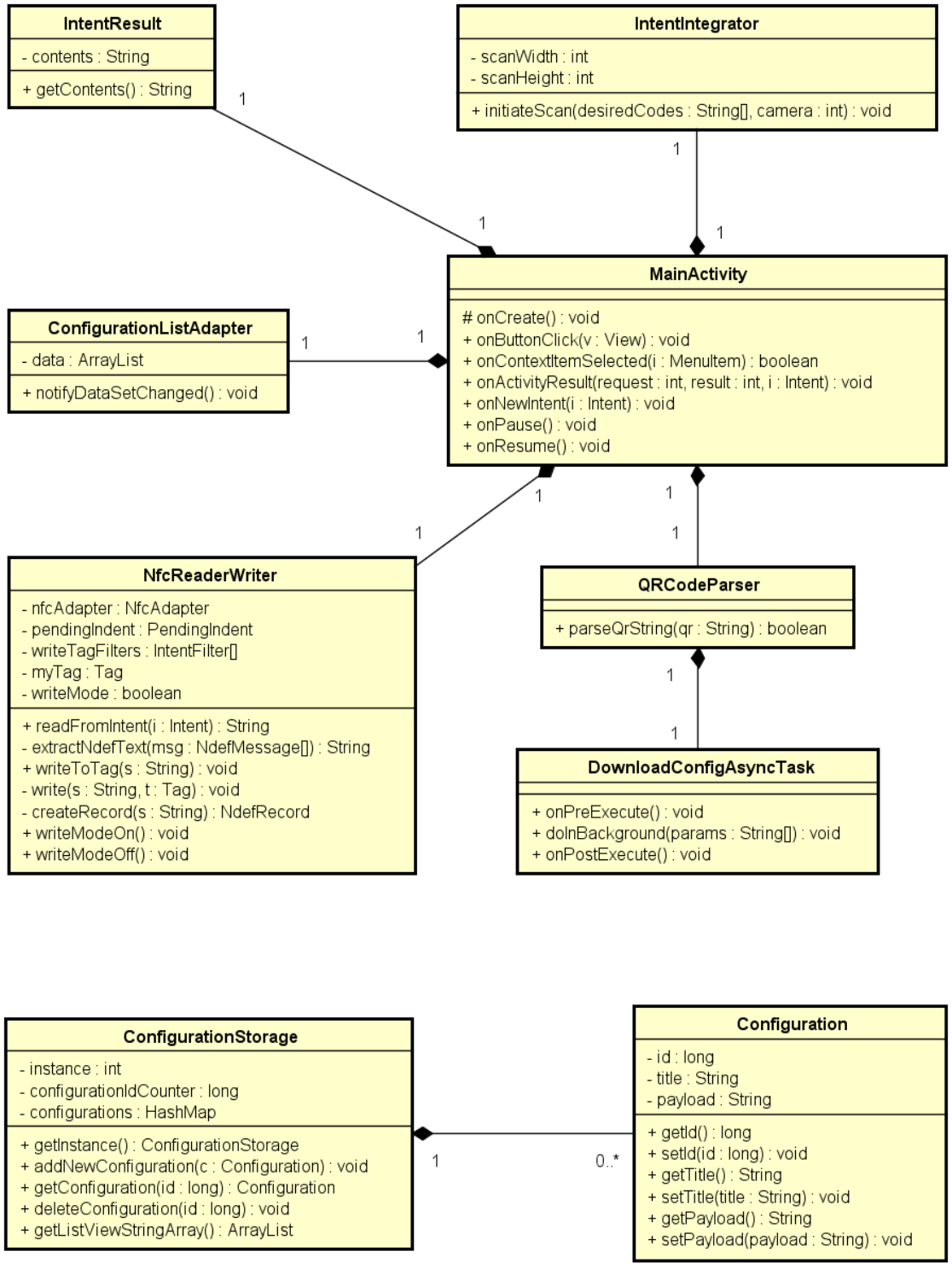


Figure 3.8: Class diagram of SECURECONFIG Android application.

3.7.2 Sequence Diagrams

The sequence diagrams shown in this section highlight how certain classes of the class diagram depicted in Figure 3.8 are created, and corresponding methods get invoked.

The sequence diagram shown in Figure 3.9 illustrates the process of loading a new configuration into the SECURECONFIG application. For this sequence diagram, a QR code that contains the configuration payload *inline* is assumed. As can be seen, the *MainActivity* passes an *IntentIntegrator* intent to the Android OS, which then invokes the third party QR code scanner (*ZXing QR Scanner)*. The returned result contains the text contained in the QR code which is then processed by the *QRCodeParser*. After successfully parsing the text, a new *Configuration* is created.

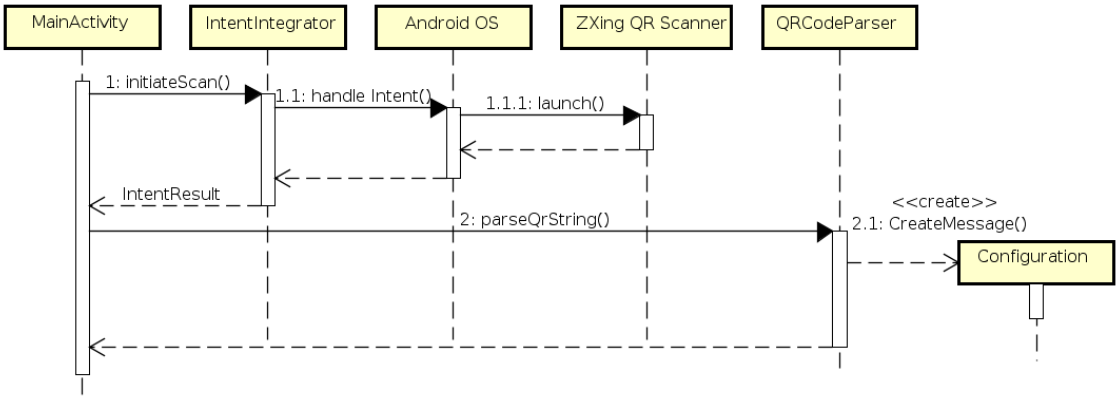


Figure 3.9: Sequence Diagram illustrating the invocation of the third party QR scanner application. In this sequence diagram, an inline payload is assumed.

For QR codes that do not contain an *inline* configuration, the sequence diagram shown in Figure 3.10 illustrates the downloading of a new configuration payload. As the process of invoking the QR code scanner is the same as for the sequence diagram shown in Figure 3.9, this process is omitted here. After the QR code was successfully detected, the received text is parsed by the *QRCodeParser*. Upon detection of an *URL* type, the asynchronous task *DownloadConfigAsyncTask* is started which downloads the content in a background thread and also creates the *Configuration* after the download finished.

The sequence diagram depicted in Figure 3.11 illustrates the process of transferring a configuration to a previously detected tag. After registering for the event, the Android operating system notifies the *SecureConfig* application of a newly detected tag. Immediately after detecting the tag, meta information such as the maximum transmission size (see Section 3.4) are read from the tag. After that, the writing process is started.

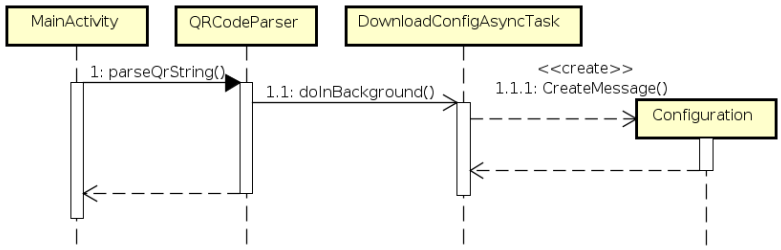


Figure 3.10: Sequence Diagram illustrating the download process of a configuration's payload from the backend.

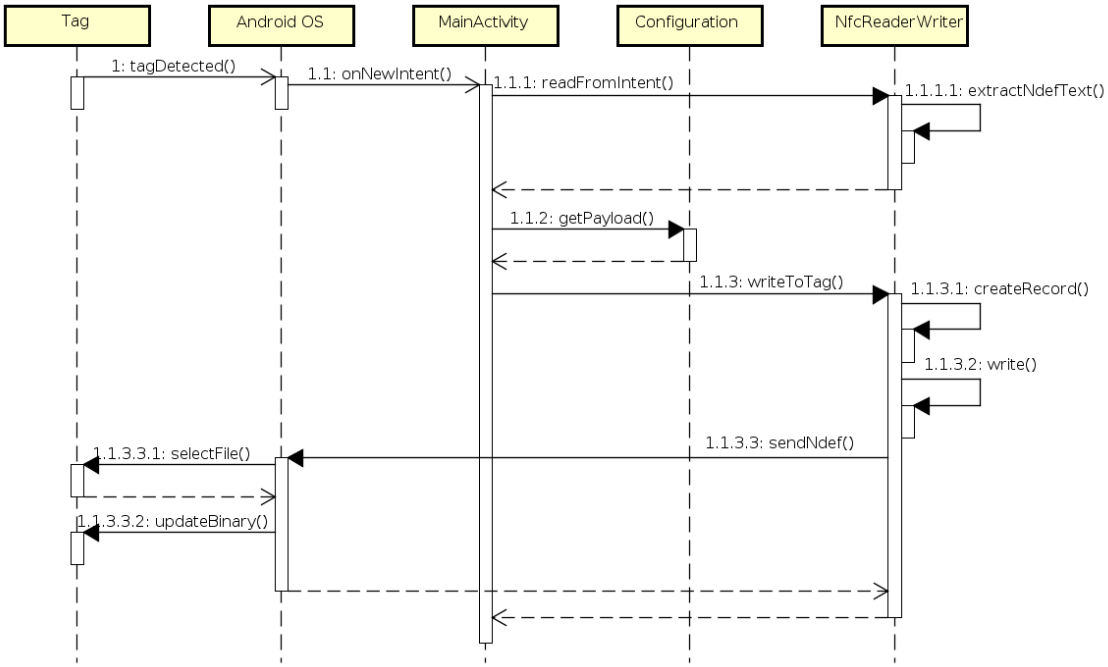


Figure 3.11: Sequence Diagram illustrating the transmission of a configuration payload to a previously detected tag using NDEF messages.

3.8 Secure Element Design

For the application implemented on the secure element, the design phase yielded a component diagram as well as a state machine which are shown in this section.

3.8.1 Component Diagram

As the application for the secure element was written in C, no class diagram can be presented. However, the source code still is grouped into logical units. Therefore, those parts are illustrated as a component diagram depicted in Figure 3.12.

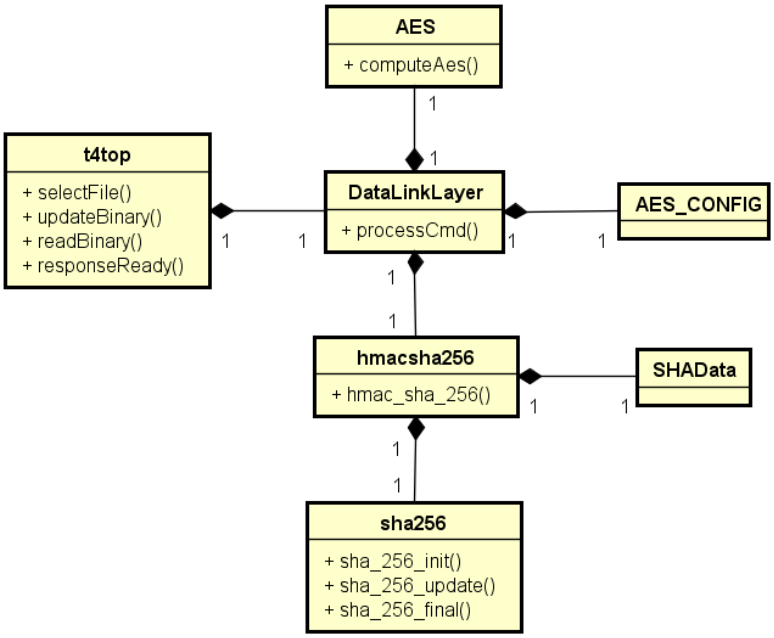


Figure 3.12: Component diagram of relevant parts of the secure element software.

As the software consists of many components which are necessary for it to run on the secure element such as command handlers or nonvolatile memory handlers, only the components directly involved in the configuration update process are shown. The tasks and responsibilities of each depicted component are discussed in the following list:

t4top: In this component, the NFC Type 4 Tag Operations (Section 3.4 are implemented. In addition to the three discussed functions *selectFile, readBinary*, and *updateBinary*, also a function *responseReady* is implemented. This function is used to signal that a response to a *C-APDU* is ready to be transmitted.

DataLinkLayer: After an NDEF message was successfully received by the t4top component, it passes the content to the DataLinkLayer component where the command is processed. In this component, the data is encrypted and verified using authenticated encryption.

AES_CONFIG: This component actually is a struct which contains all data necessary to perform AES128 encryptions/decryptions. It contains the data to be en-/decrypted, the private key, and information such as the block cipher mode of operation.

AES: This component provides the AES en-/decrypt functionality. The component offers an easy to use interface to the AES functionality which is implemented in hardware on the secure element.

hmacsha256: The hmacsha256 component offers an interface to compute a keyed-hash message authentication code based on the SHA256 hash algorithm.

SHAData: This component actually is a struct on which the SHA256 hash algorithm operates and stores the final result.

sha256: This component is an implementation of the SHA256 hash algorithm. As no hardware support for this hash algorithm exists at the used security controller, this is a pure software implementation.

3.8.2 NDEF State Machine

The NFC Type 4 Tag Operations (Section 3.4) need to be sent in a specific sequence such that an NDEF message is detected. Therefore, the *t4top* component discussed earlier contains a state machine which is depicted in Figure 3.13.

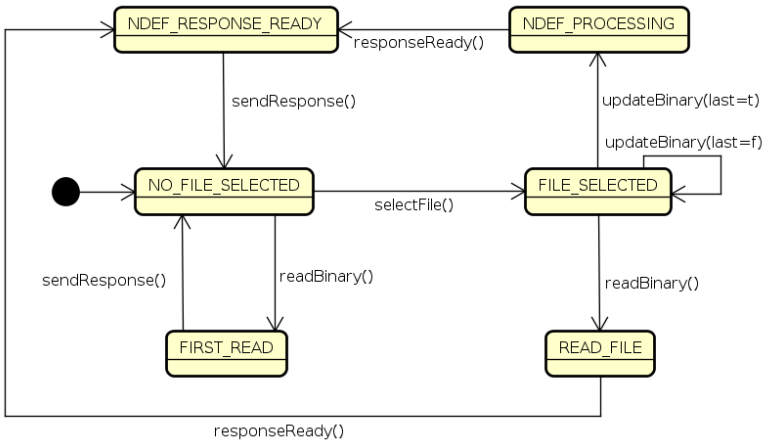


Figure 3.13: State machine regarding the different states when implementing the NFC Type 4 Tag Operations.

As can be seen in the state machine, a read operation without previously selecting any file results in the *FIRST_READ* state in which the capability container file is returned. If a file is selected, it can either be read (*READ_File*) or updated. In the case of a write operation, multiple *updateBinary* C-APDUs might be sent if a message is split into multiple packages. When detecting the final package, the NDEF message is processed (*NDEF_PROCESSING*). In any case, after sending a response (*NDEF_RESPONSE_READY*) the state machine returns to the initial *NO_FILE_SELECTED* state.

4 Implementation

In this chapter, the tools used during development as well as the involved platforms and included libraries are going to be presented. Also, the necessary tools and equipment for simulation and emulation of system parts during development are going to be discussed in this chapter. After that, implementation details and code snippets for all three involved components of the system are shown to highlight important aspects of the implementation.

4.1 Libraries and Third Party Components

The three components of the system, shown in Figure 3.5, will be implemented and executed on different platforms, thus requiring the use of multiple programming languages. Therefore, the development tools and libraries used to develop each system component will be discussed separately for each element.

4.1.1 Mobile Device

The device which is used to transport configurations from the backend to the secure element can be an arbitrary device which fulfils the following requirements:

1. **NFC Interface**: The device needs to be equipped with an NFC interface to transfer data to the secure element.
2. **Camera**: A camera is needed to be able to read configuration QR tags.

3. **Network Interface**: If configuration data should be fetched from the backend, also a network interface is needed for the data transfer.

For the prototypical implementation of the data transfer application, a smartphone was chosen because the necessary hardware is present in nearly all current smartphones. Because of the openness of the platform, Android was selected. The Android platform and some of its security features were already discussed in Section 2.1.6. During this master's thesis, a Google Nexus smartphone was used for implementation and testing.

Java: Native applications for the Android operating system are mainly developed using Java. Java offers all features of modern programming languages such as object-orientation or concurrency. To be executed on Android, Java code is compiled to bytecode which is then translated into DEX bytecode. This DEX bytecode is then executed in the Dalvik virtual machine on Android which offers some security-related advantages, such as executing applications in a Sandbox [EOMC11].

Android Studio: The official IDE which should be used for application development on Android is called Android Studio. The integrated development environment is based on the proprietary *IntelliJ*, however, Android Studio is freely available. The IDE offers many features related to Android application development such as a drag and drop user interface designer or templates to create Android applications more easily. Android Studio also includes a virtual machine where Android and newly developed applications can be simulated. Also, applications can be executed and debugged directly on Android smartphones with enabled *developer mode* and *USB debugging* modes.

Gradle: To automate the build process, *Gradle* is used as default tool in combination with Android Studio. In a configuration file, tasks such as *compile* or *run* can be specified. For all tasks, dependencies can be defined. Gradle then uses a DAG (directed acyclic graph) to determine the order of execution. Dependencies of the implemented application can be listed in the *build.gradle* configuration file which are then fetched automatically. Also, Gradle supports incremental builds, meaning only updated parts of the calculated tree are executed.

ZXing: The library *ZXing* (zebra crossing) was used to provide QR code reading capabilities to the implemented application. For the library to function properly, the application *Barcode Scanner* needs to be installed on the Android smartphone. When using ZXing, an intent (*IntentIntegrator*, see Figure 3.8) is posted by the application which invokes the standalone QR code scanner. If a code was detected successfully, the resulting text is returned to the calling application. The library is released under the *Apache license* and therefore is open source.

4.1.2 Backend

The backend which creates and stores configurations, creates QR codes and provides potential downloads for configuration payloads is implemented as a web application. To post no requirements regarding the backend's hardware, JavaScript was used to develop the application. Thus, the backend can be executed also on mobile devices such as Android.

JavaScript: The programming language used to implement the backend, *JavaScript*, is an interpreted language that is also untyped. JavaScript supports object-oriented concepts as well as functional programming. Together with HTML and CSS, it is seen as a core technology of the world wide web. Similar to Java, JavaScript code is executed in virtual machines. Historically, JavaScript was seen as an interpreted language used solely for web development. However, nowadays often *just-in-time compilation* is applied, which allows using JavaScript for domains such as desktop application development.

NodeJS: The runtime environment used to execute the backend implementation, *NodeJS* is a JavaScript runtime environment based on Googles JavaScript engine. NodeJS is open-source and cross-platform with many modules of it written in JavaScript. Also, new modules can be implemented using the JavaScript programming language. NodeJS is capable of asynchronous I/O which is perfect for implementing web applications. A major advantage when compared to other technologies is the fact that NodeJS can be installed on Android, which allows using mobile devices such as smartphones or tables as backends in the prototype developed in this master's thesis.

WebStorm: The integrated development environment (IDE) used to implement the backend, *WebStorm*, which is offered by the producers of *IntelliJ*. WebStorm offers features such as syntax highlighting and code completion when writing JavaScript code. It also has an integration for NodeJS which makes application development using that runtime environment very easy.

Express: To simplify the web application development, *Express* was used which is a NodeJS library for the development of web applications. The library offers lightweight layers which use existing NodeJS functionality to provide an API for most used web functionalities such as providing files or offering *HTTPS* support.

pem: To create or import private keys and certificates (which are needed for HTTPS), a library named *pem* is used. By integrating this library, self-signed certificates can be created at startup which are then directly used to enable HTTPS connections to the backend. Self-signed certificates are used in the prototype implemented in this master's thesis because no fixed URL for which a certificate could be ordered existed.

qr-image: To create QR codes based on the data structure presented in Section 3.3.1, the library *qr-image* is used. The library can generate QR codes in many formats such as png or pdf without requiring additional dependencies.

CryptoJS: For all cryptographic functions such as *AES* or *HMAC-SHA256*, the library *CryptoJS* which is provided by Google was used. CryptoJS offers implementations of most cryptographic algorithms in JavaScript using best practices and patterns. CryptoJS uses specific data types to represent keys and ciphertexts; therefore it also includes tools to export that data to various formats such as an *OpenSSL* compatible form. This library also has no further dependencies.

4.1.3 Secure Element

The secure element which is used in the prototype implemented in this master's thesis was already discussed in Section 1.2. As this is a specialized hardware provided by *Infineon*, the programming language, development tools and libraries were given.

C89: The programming language used to implement features on the secure element provided by *Infineon* is C89, which is one of the most well-known programming languages. The source code is compiled into an executable, which can then be flashed to the secure element hardware.

μVision: The integrated development environment used to implement functionality on the secure element was μVision by Keil. The IDE offers powerful features for development of embedded software such as the possibility to view and debug the assembler code generated from the written C source code. If implementing for a specific hardware such as the secure element used in this master's thesis, it is possible to simulate the hardware or use an emulator when debugging. Both methods are going to be described in Section 4.3. The toolchain to compile, simulate and debug the written code was provided by *Infineon*.

Card Emulator: To simulate a mobile device which sends NFC commands to the secure element's NFC module, a card emulator was used. To be able to control the card emulator, a Reader API, and an *Infineon* internal software (where packets can be specified and sent at byte level) are used.

FPGA Emulator: A Hitex FPGA (field programmable gate array) emulator was used to emulate the secure elements hardware. The complete setup of the debugging process, including a detailed description of the FPGA emulator, will be discussed in Section 4.3.

4.2 Development Toolchain

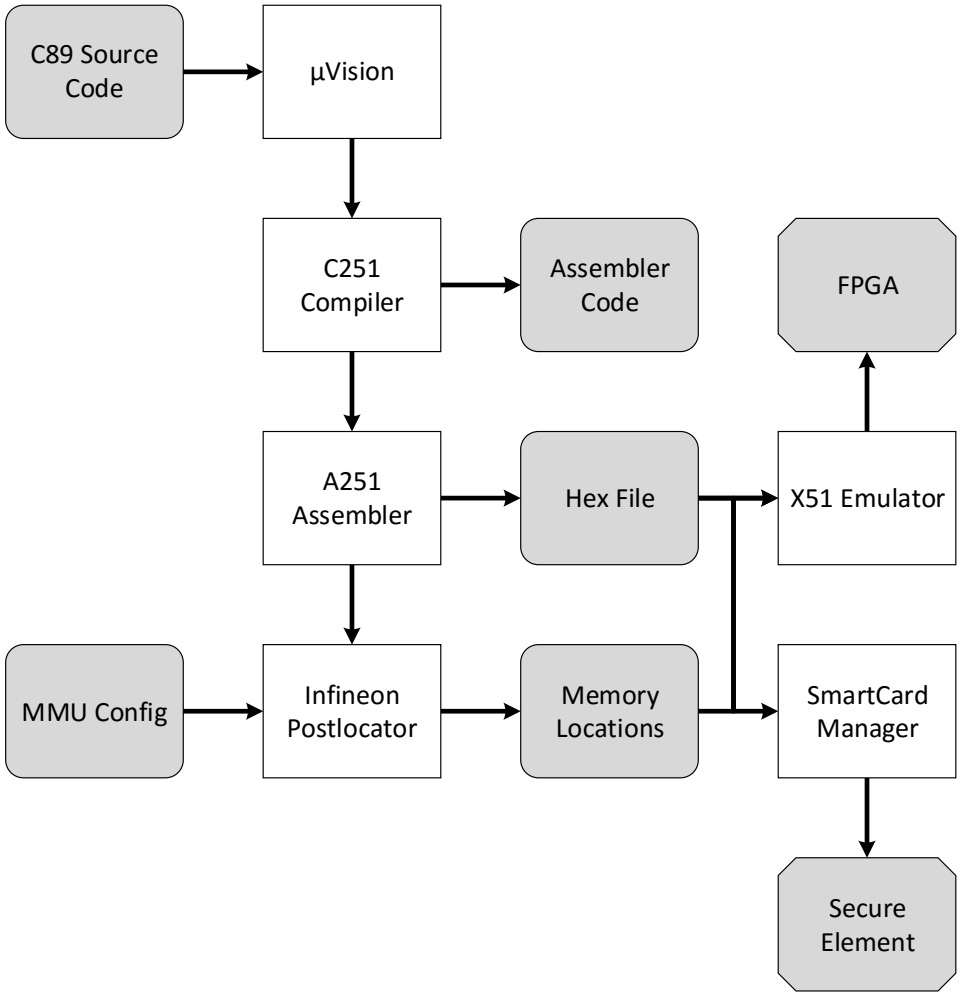


Figure 4.1: The software toolchain used for developing the secure element software.

The software toolchain used during development of the secure elements software components is depicted in Figure 4.1. As can be seen there, various tools are used to generate all necessary artifacts. The C89 based source code is written in μVision which was already introduced in Section 4.1.3. The included C251 compiler generates assembler code, which is used by the also included A251 assembler to generate a hex file. Using the provided MMU configuration, the Infineon postlocator then generates the memory locations for the respective hardware configuration. The generated hex file and memory locations can then be used by the X51 emulator which is integrated into μVision to debug the application on an FPGA emulator. Also, the generated files can be flashed to a secure element using a software called SmartCardManager.

4.3 Debugging Environment

To be able to test and debug the implemented features, an FPGA emulator (Hitex Tanto 2) is used. To emulate the correct hardware, a processor-netlist needs to be loaded first. After the hardware is set up, the application can be loaded onto the emulated hardware. When using μVision, the compiled file is loaded directly from the IDE. To use NFC functionality, an analog front end (AFE) is connected to the FPGA. This AFE is then connected with a REFPICC card-interface. This interface acts as an NFC tag; therefore it is possible for NFC readers to communicate with it. Figure 4.2 illustrates the whole debugging setup. As can be seen, there are two possibilities to communicate with the REFPICC card-interface. Either a mobile device or a card reader can be used to send NFC commands to the card interface.

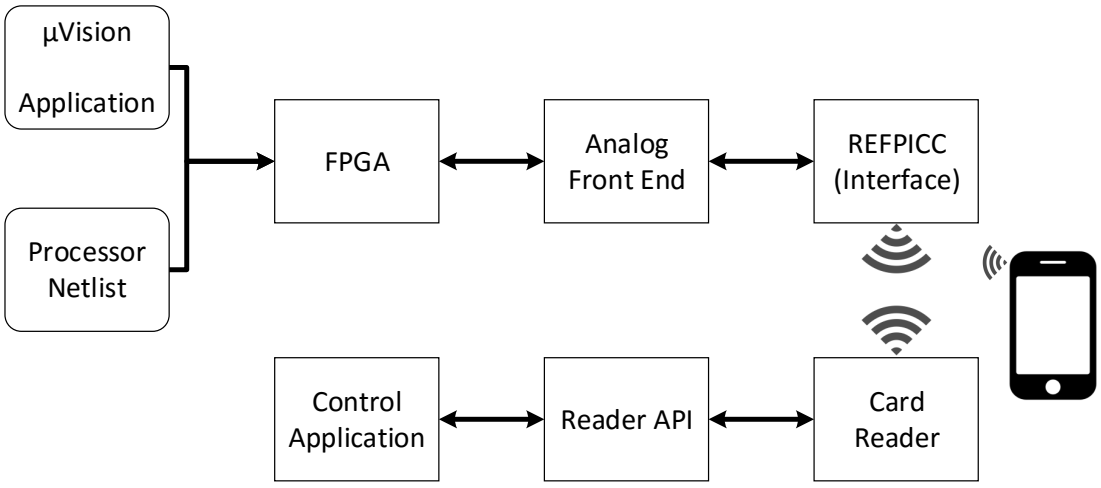
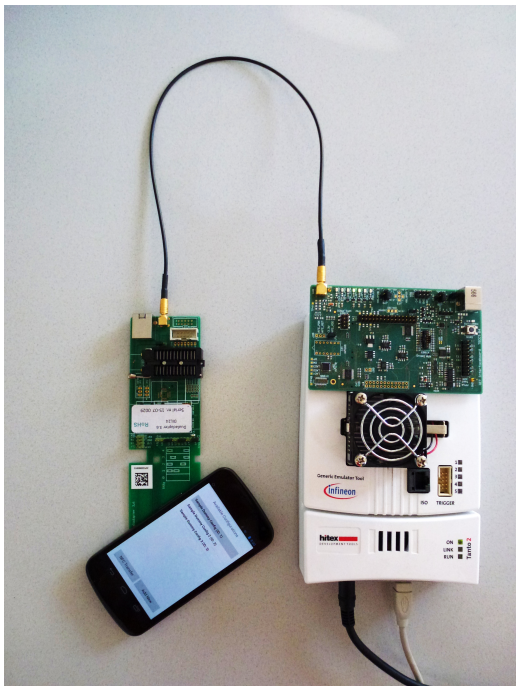


Figure 4.2: Debugging setup using a FPGA emulator. To send NFC commands to the emulator, either a smartphone or a card reader can be used.

When using the card reader, a so-called *reader API*, which is used in combination with an *Infineon* internal software, can be used to communicate with the REFPICC interface. Using this software, it is possible to compose arbitrary APDU packets and send them to the REFPICC interface. Figure 4.3 shows the debugging setup when using a smartphone to transfer data and when using a card reader.



(a) FPGA simulator with attached REFPICC interface and a smartphone sending NFC commands.

(b) REFPICC interface used in combination with the card-reader to transfer data via NFC.

Figure 4.3: Debugging environment using FPGA emulator, card-interface, smartphone, and card-reader. The FPGA emulator is connected to μVision to allow debugging of the deployed application code.

4.4 Hardware Prototype

The hardware prototype that was used to verify that the implemented software components work as expected on real hardware was developed by Lesjak et al. [LRB+14b]. The hardware module named *CUTIN* uses an Infineon XMC 4500 as host controller on an evaluation board. The host controller offers connection interfaces such as USB and Ethernet, as also depicted in Figure 1.3. This host controller is connected to the security controller via I2C. The security controller integrated into the CUTIN module is an EAL5+ (Evaluation Assurance Level 5+ [MFMP07]) certified security controller by Infineon. This controller provides security features such as secure data storage and code execution by using a self-checking dual CPU concept, integrity checks for data transfers and caches and encrypted memory and calculations in the CPU according to Lesjak. An NFC antenna necessary to communicate with the device while also powering it via the NFC field is integrated into the CUTIN module. The hardware prototype is depicted in Figure 4.4.

Figure 4.4: The CUTIN hardware prototype comprises a host controller, security controller and an NFC antenna on an Infineon evaluation board.

4.5 Code Samples of Important Concepts

In this section, code samples of all important concepts which were implemented in this master's thesis are shown. The functionality and possibly used libraries for each presented code snippet will also be discussed.

4.5.1 NFC Type 4 Tag Operations

The NFC Type 4 Tag Operations that were discussed in Section 3.4 were implemented on the secure element to allow mobile devices to communicate with the tag using NDEF messages. The implementation was adapted from an earlier version already implemented on the hardware. A code snippet handling *Select* can be seen in Listing 4.1. There, the principle structure, already shown in Table 3.4, is verified and the file identifier is extracted from the data field. As can be seen in the code, only two files are supported: the capability container and a file *E104* which is specified as being mandatory for NDEF communication. If any other file identifier is provided, the error code *0x6A82* will be returned.

```
1 // see if select command
2 if (pbAppDataRx[APDU_P1] == 0x00
3   && (pbAppDataRx[APDU_P2] == 0x0C
4     || pbAppDataRx[APDU_P2] == 0x00)
```

```
5     && pbAppDataRx[APDU_P3] == 0x02)
6   {
7     // select fileId based on APDU_DATA parameter
8     UINT16 *fileId = (UINT16*) &(pbAppDataRx[APDU_DATA]);
9
10    // only CC and mandatory NDEF file 0xE104 are supported
11    if ((*fileId) == FILE_ID_CC || (*fileId) == 0xE104)
12    {
13      selectedFile = *fileId;
14      *(UINT16*)(*ppbAppDataTx) = ISOSW_SUCCESS;
15    }
16    else
17    {
18      // file id not found
19      selectedFile = 1;
20      *(UINT16*)(*ppbAppDataTx) = 0x6A82;
21    }
22  }
```

Listing 4.1: Secure Element: Select command which supports CC and mandatory NDEF file 0xE104.

The code sample shown in Listing 4.2 demonstrates the handling of the *ReadBinary* command. Again, only two files (capability container and E104) are handled here. As can be seen in the code snippet, the 2-byte *offset* and the *data length* which will be read are extracted from the parameters. The file is then read starting from the provided offset, and the specified number of bytes is returned. Depending on the file size, read access to a file may be handled by several subsequent *ReadBinary* commands.

```
1  if (selectedFile == 0x0000 || selectedFile == 0x0001)
2  {
3    // no application or file was selected, do not return any data
4    *(UINT16*)((*ppbAppDataTx) + dataLength) = ISOSW_NOT_ALLOWED;
5    return CMDHANDLER_PASS;
6  }
7
8  // read offset and datalength from C-APDU
9  dataOffset += pbAppDataRx[APDU_P2];
10 dataOffset += pbAppDataRx[APDU_P1] << 8;
11 dataLength  = pbAppDataRx[APDU_P3];
12
13 switch (selectedFile)
14 {
15   case FILE_ID_CC:
16     // capability container, reply the corresponding file
17     file = nvm->bEEDATA_FILE_E103; break;
18   case FILE_ID_NDEF:
19     switch (ndefState) {
20       // according to ndefState, either reply
21       // mandatory NDEF file or no response
```

```
22        case NDEF_STATE_FIRST_READ:
23          file = nvm->bEEDATA_FILE_E104; break;
24        case NDEF_STATE_PROCESSING:
25          file = clTxNoResponseYet; break;
26        case NDEF_STATE_HAS_RESPONSE:
27          file = clTxNdefFile; break;
28        case NDEF_STATE_WAIT_FOR_REQ:
29          file = clTxNoResponseYet; break;
30        default:
31          file = nvm->bEEDATA_FILE_E104; break;
32      }
33      break;
34      default:  break; // should not reach
35    }
36
37    // copy dataLength bytes of file content starting from offset
38    memcpy(*ppbAppDataTx, file + dataOffset, dataLength);
39    *(UINT16*)((*ppbAppDataTx) + dataLength) = ISOSW_SUCCESS;
40    *pwAppDataTxLen += dataLength;
41  }
```

Listing 4.2: Secure Element: ReadBinary method which selects the appropriate file according to the Select command and replies the content starting from offset with a maximum length which is specified in the C-APDU.

The third Type 4 Tag Operation, *UpdateBinary*, is demonstrated in the code snippet shown in Listing 4.3. Again, *offset* and *data length* parameters need to be handled as an NDEF message might be split into several *UpdateBinary* C-APDUs depending on its size. All received parts are stored in a nonvolatile memory until the data transmission is finished. In that case, the handling of the received command and its data is invoked.

```
1  // read offset and datalength from C-APDU
2  dataOffset += pbAppDataRx[APDU_P2];
3  dataOffset += pbAppDataRx[APDU_P1] << 8;
4  dataLen     = pbAppDataRx[APDU_P3];
5
6  pbAppDataRx += (wAppDataRxLen - dataLen);
7
8  // copy the part of the Ndef message into non volatile file
9  if (StoreNdefPartInNV(file, dataOffset, pbAppDataRx, dataLen)
10     != T4TOP_OK)
11 {
12   *(UINT16*)(*ppbAppDataTx) = 0x6A80;
13   return !CMDHANDLER_PASS;
14 }
15
16 // check if last write operation of a split NDEF message
17 // case 1: only NLEN is written with a value > zero
18 if (dataLen == 0x02 && dataOffset == 0x00
19     && *((UINT16*) pbAppDataRx) > 0x00)
```

```
20 {
21   ndefState = NDEF_STATE_PROCESSING;
22   DataLinkLayer_ProcessCmd(ORIGIN_CL, &file[5], file[4]);
23 }
24 // case 2: the complete NDEF message was sent in a single APDU
25 else if (dataLen > 0x02 && dataOffset == 0x00
26          && *((UINT16*) pbAppDataRx) == (dataLen - 2))
27 {
28   ndefState = NDEF_STATE_PROCESSING;
29   DataLinkLayer_ProcessCmd(ORIGIN_CL, &file[5], file[4]);
30 }
```

Listing 4.3: Secure Element: UpdateBinary command. Two cases need to be handled here. Either an NDEF message is sent in a single C-APDU or the message is split into multiple C-APDUs.

If all three Type 4 Tag Operations are implemented on the tag, Android allows sending NDEF messages to such tags. The corresponding code can be seen in the code snippet depicted in Listing 4.4. As can be seen there, for each NDEF message sent a single NDEF record of type `TNF_WELL_KNOWN` is sent. This indicates that the second identifier, in this case, `RTD_TEXT` further specifies the payload's type. NDEF messages can be sent to already discovered tags, to which a connection needs to be established and closed after sending.

```
1  private void write(byte[] payload, Tag tag)
2    throws IOException, FormatException
3  {
4    // create single NDEF record to transmit
5    // add it to a new NDEF message
6    NdefRecord recordNFC = new NdefRecord(
7      NdefRecord.TNF_WELL_KNOWN,  NdefRecord.RTD_TEXT,
8      new byte[0], payload);
9    NdefRecord[] records = { recordNFC };
10   NdefMessage message = new NdefMessage(records);
11
12   // get NDEF instance for the discovered tag
13   Ndef ndef = Ndef.get(tag);
14   // connect to tag to enable read/write
15   ndef.connect();
16   // send the prepared NDEF message to the tag
17   ndef.writeNdefMessage(message);
18   // close the connection after sending
19   ndef.close();
20 }
```

Listing 4.4: Mobile Device (Android): Sending an NDEF message to an already discovered tag.

To detect tags, applications need to register a callback that is invoked whenever the desired action is detected by Android OS. As can be seen in Listing 4.5, the event that application needs to register for is `ACTION_TAG_DISCOVERED`.

```
pendingIndent = PendingIntent.getActivity(
  mainActivity, 0, new Intent(mainActivity,
  mainActivity.getClass()).addFlags(
    Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
IntentFilter tagDetected = new IntentFilter(
  NfcAdapter.ACTION_TAG_DISCOVERED);
tagDetected.addCategory(Intent.CATEGORY_DEFAULT);
writeTagFilters = new IntentFilter[] { tagDetected };
```

Listing 4.5: Mobile Device (Android): Registering in Android OS to be notified of newly detected tags.

As mentioned before, to detect tags the application needs to register for the event of a newly detected tag. As soon as the tag is detected, the callback shown in Listing 4.6 is invoked. In addition to storing the detected tag for possible future write operations, also the tag information is read in this step. Thus, the method shown in this listing is sufficient to read NDEF messages from detected tags.

```
public String readFromIntent(Intent intent)
{
  if (NfcAdapter.ACTION_TAG_DISCOVERED
    .equals(intent.getAction()))
  {
    tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
  }

  String action = intent.getAction();
  if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)
    || NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)
    || NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action))
  {
    Parcelable[] rawMsgs = intent
      .getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
    NdefMessage[] msgs = null;
    if (rawMsgs != null)
    {
      msgs = new NdefMessage[rawMsgs.length];
      for (int i = 0; i < rawMsgs.length; i++)
      {
        msgs[i] = (NdefMessage) rawMsgs[i];
      }
    }
```

```
25        return extractNdefText(msgs);
26      }
27      return null;
28    }
```

Listing 4.6: Tag is discovered by Android OS and an intent is passed to the application which is then responsible to read NDEF content.

4.5.2 Authenticated Encryption

Authenticated Encryption is implemented at the backend where the configuration payload is encrypted, and a MAC is calculated, as well as on the secure element where the encrypted data is decrypted, and the MAC is verified. The principle that was implemented is depicted in Figure 3.6. The code snippet depicted in Listing 4.7 shows the encryption and calculation of the message authentication code in JavaScript. As discussed in Section 4.1.2, the library *CryptoJS* is used for all cryptographic functions. The implementation at the backend is similar to this one, the decryption and MAC calculation are done by using libraries provided by *Infineon*.

```
1  var CryptoJS = require("crypto-js");
2  var AES = require("crypto-js/aes");
3  var HMACSHA256 = require("crypto-js/hmac-sha256");
4
5  // generate nonce, iv and load AES key
6  var nonce = generateNonce();
7  var iv = generateIv();
8  var key = loadAesKey();
9  var plaintext = nonce + text;
10
11 // encrypt nonce+plaintext
12 var encrypted = CryptoJS.AES.encrypt(plaintext, key,
13   {iv: iv, mode: CryptoJS.mode.CBC});
14 var cipherlen = Math.ceil(text.length*2/16)*16;
15 var hexcipher = CryptoJS.enc.Hex.stringify(encrypted.ciphertext)
16   .substring(0, cipherlen);
17
18 // calculate MAC from ciphertext + nonce
19 var MAC = HMACSHA256(encrypted, nonce);
20 var hexMAC = CryptoJS.enc.Hex.stringify(MAC);
21
22 // concatenate ciphertext + MAC
23 var result = hexcipher + hexMAC;
```

Listing 4.7: Implementation of Encrypt-then-MAC as depicted in Figure 3.6.

The libraries available for the secure element did not include implementations of an HMAC algorithm (as discussed in Section 2.1.5). Therefore, *HMAC-SHA256* was imple-

mented using an existing implementation of the SHA256 algorithm. The corresponding code sample can be seen in Listing 4.8.

```
1  // block size in SHA256 is 64 byte, so check key_len
2  if (key_len > 64) return -1;
3  // if key is too short, pad it with zeros
4  memcpy(&padded_key[64-key_len], key, key_len);
5  // XOR padded key with 0x36
6  for (i=0; i < 64; ++i)
7  {
8    work[i] = padded_key[i]^0x36;
9  }
10 // append message to that and apply SHA256
11 sha_256_init(buffer);
12 sha_256_update(buffer, work, 64);
13 sha_256_update(buffer, message, len);
14 sha_256_final(buffer);
15 memcpy(intermediate, buffer, 32);
16 // XOR padded key with 0x5c
17 for (i=0; i < 64; ++i)
18 {
19   work[i] = padded_key[i]^0x5c;
20 }
21 // append previously calculated hash to that and apply SHA256
22 sha_256_init(buffer);
23 sha_256_update(buffer, work, 64);
24 sha_256_update(buffer, intermediate, 32);
25
26 // final result in buffer
27 sha_256_final(buffer);
```

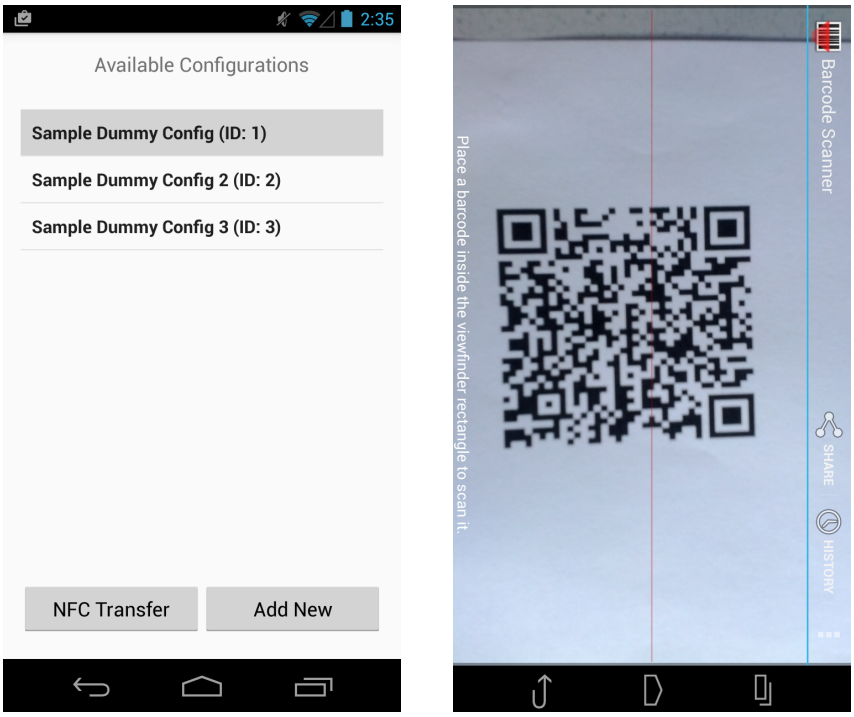
Listing 4.8: Implementation of HMAC-SHA256 using an existing implementation of the SHA256 hash algorithm.

4.6 SECURECONFIG User Interface

As already mentioned in Section 3.2, a mobile device is used to transport configuration payloads from a backend to the secure element. For simplicity reasons, in this master's thesis, an Android smartphone is used as a mobile device, as discussed in Section 4.1.1. This section presents the implemented user interface of the so-called SECURECONFIG application.

The screenshots shown in Figure 4.5 illustrate the user interface of the SECURECONFIG application. Figure 4.5(a) illustrates the *main screen* of the application. In this case, already three configurations were loaded into the application. By clicking the *Add New* button, the QR scanner is launched to read a new configuration. This is shown in Fig-

ure 4.5(b). When long clicking an already existing application, a context menu shown appears where configurations can be deleted, or details of the selected configuration can be shown.



(a) Main Screen. (b) QR code scanning.

Figure 4.5: Screenshots demonstrating the user interface of the SECURECONFIG application.

5 System Evaluation

In this chapter, the proposed system is going to be evaluated. First, to highlight all assets and potential threats to them, a thorough security and risk analysis [MLY05, SS04] on the implemented prototype will be done. In this analysis, countermeasures for the found threats and attacks as well as potential residual risks are listed. The second part of this evaluation will analyze the required packet size and compare it to an unsecured transport of data.

5.1 Security and Risk Analysis

Threat modeling is a critical concept when trying to evaluate the security of a system, as Myagmar et al. [MLY05] state:

> *Prior to claiming the security of a system, it is important to identify the threats to the system in question. Enumerating the threats to a system helps system architects develop realistic and meaningful security requirements.*

The authors highlight that the process of system security engineering is an iterative one, where all three phases (*thread modelling, specifying security requirements,* and *developing security mechanisms* are repeated until all listed threads are mitigated or declared as non-amendable (Figure 5.1).

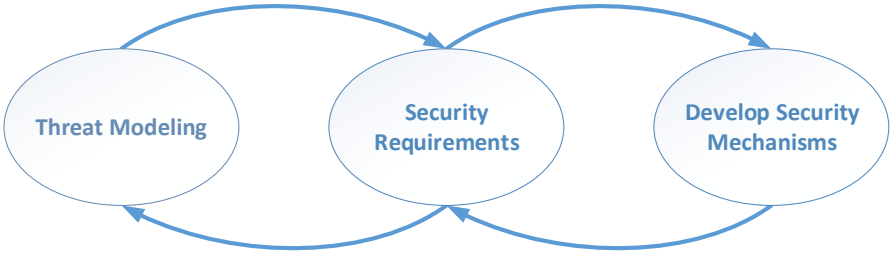


Figure 5.1: Process of system security engineering, according to Myagmar et al. [MLY05].

5.1.1 Entities

As the first step for the conducted security and risk analysis, the involved **entities (E)** are identified. In addition to identifying all entities, **assumptions (AS)** (numbered for each entity) regarding them are made as well.

(E1) Secure Element Vendor: The vendor of the *secure element* is providing the components to equip devices with a secure NFC-configuration interface. Those components include the *secure element* as well as the necessary software for the configuration process (software for *backend, mobile device, secure element*). The secure element vendor also deploys an initial encryption/decryption key at the secure element.

(E2) Device Vendor: The device vendor integrates the preconfigured *secure element* into a hardware solution such as a smart sensor or a smart home device. When integrating the secure element, the device vendor has physical access to potential (debug-) interfaces of the *secure element* and the hardware in general. The device vendor also might change the initial key deployed by the *secure element vendor*.

(E3) Customer: The customer is using the combined hardware solution as provided by the *device vendor*. Customers can either be business customers or private customers. Physical access to the hardware and potentially the *secure element* is possible by the customer.

(E4) Configuration Changer: The person changing a configuration at the hardware can either be the *customer* itself or (most likely in the case of a business customer) a third party such as a maintenance worker. Also, the person changing configurations has physical access to the hardware.

(E5) Adversary: A potential adversary might try to attack the system either by utilizing the wireless interface or by trying to gain physical access to the hardware. All three components of the system (*mobile device, secure element, backend*) might be subjected to potential attacks.

(E6) Mobile Device: The mobile device is used by the configuration changer to transport configuration payload from the *backend* to the receiving *secure element* and to deploy the configuration using the NFC interface. An adversary might be able to get physical access to such a mobile device as well.

(E7) Secure Element: The secure element is integrated into some hardware solution. It is responsible to maintain the key material and to decrypt and verify new configuration data.

(E8) Backend: The backend is used to create and manage configuration packages. The configurations and the corresponding key material for the encryption of the data are stored in the backend.

5.1.2 Assumptions

Regarding the previously identified entities, some assumptions are made. The assumptions specify the trustworthiness of some entities as well as the security properties of some components.

(AS1): The secure element vendor is assumed to be trustworthy.

(AS2): The device vendor is assumed to be honest but curious.

(AS3): The customer is assumed to be honest but curious.

(AS4): The configuration changer is assumed not to be trustworthy.

(AS5): The adversary is assumed to be able to conduct online and physical attacks.

(AS6): The mobile device is assumed to offer the possibility to be equipped with a SE.

(AS7): It is assumed that all physical attacks regarding the secure element are infeasible.

(AS8): The backend is assumed to be secured against physical attacks.

5.1.3 Assets

After identifying all involved entities and corresponding assumptions, the **assets (A)** which need to be protected are determined.

(A1) Configuration Data: The configuration data that is transferred via the NFC-based configuration interface proposed in this master's thesis is considered as the primary asset. Configuration data might include private keys or sensitive configuration data and therefore is considered as confidential. All three components of the system (*mobile device, secure element, backend*) access the configuration data. Therefore, security measures need to be considered for all involved parts.

(A2) Key Material: The key material which is needed to encrypt the configuration data is considered as the second asset. An *adversary* who has access to the key material can easily decrypt and read the confidential configuration data that is transferred via the NFC-based configuration interface.

(A3) Device Functionality: The functionality of devices is considered as third asset which needs to be protected from adversaries. Functionality can be compromised by either deploying faulty configurations that disable or even destroy the device or by denying the device's service by using the provided NFC-based configuration interface.

5.1.4 Threats

Considering all identified entities, the corresponding assumptions and the assets which need to be protected, the system now can be reviewed concerning potential **threats (T)**. For each threat, **countermeasures (C)** and/or **residual risks (R)** are then determined and listed. The threads are grouped by the corresponding (*active*) **entities (E)** which could cause the related threats. For each threat, all potentially threatened assets are listed as well. A threat might compromise one or multiple of those listed assets.

(E1) Secure Element Vendor

(T1) The secure element vendor might unintentionally leave back-doors (either in hardware or software) such as debug interfaces open.
Potentially threatened assets: **(A1), (A2), (A3)**

(C1) As a countermeasure, the developed hardware and software components are certified by a third party regarding their security.

(T2) Cryptographic algorithms and security concepts such as authenticated encryption may not be implemented correctly.
Potentially threatened assets: **(A1), (A2), (A3)**

(C2) In addition to certifying the software components, only best practices such as *Encrypt-then-MAC* are used.

(T3) The secure element vendor might only allow keys up to a certain length which could be too short to be considered secure. Also, the supported algorithms might be already broken or not considered as secure (e.g., DES, MD5).
Potentially threatened assets: **(A1), (A3)**

(C3) Algorithms that are considered secure (such as AES or SHA-256) are used in combination with keys of a length that is shown to be infeasible to attack at the moment.

(E2) Device Vendor

(T4) The provided secure element might not be integrated correctly into the device by the device vendor. Thus, making it possible to attack the integrated secure hardware element through existing interfaces.
Potentially threatened assets: **(A1), (A2), (A3)**

(C4) To prevent wrongly integrated secure hardware elements, the device vendor also gets certified by a third party.

(T5) When integrating the secure element, the device vendor might use the provided security functionality in a wrong way. For example, cryptographic functions could be called in a not specified order.
Potentially threatened assets: **(A1), (A2), (A3)**

(C5) To prevent such threats, the provided API must not allow such incorrect calls.

(T6) As the device vendor can change the initial key specified by the secure element vendor, it is possible that keys are changed such that they become insecure. For example, the same key could be used for multiple devices or a key could be chosen such that it can be derived from device parameters such as the name of the device.
Potentially threatened assets: **(A1), (A2), (A3)**

(R6) As changing the keys is the device vendor's responsibility, no countermeasure can be taken to prevent the deployment of insecure keys.

(E3) Customer

(T7) Also customers are allowed to change keys, which results in the same threat as threat T5.
Potentially threatened assets: **(A1), (A2), (A3)**

(R7) As for T5, also for T6 no countermeasure is possible as customer's can change keys to arbitrary values.

(T8) As the devices are deployed in a business customer's premises or anywhere a private customer wishes, it might be possible that the customer does not prevent physical access to the hardware components. Attackers therefore could perform any kind of attack that requires physical access to the hardware.
Potentially threatened assets: **(A1), (A2), (A3)**

(C8) All components (mobile device, secure element, backend) are designed in such a way that gaining physical access to them does not harm the security of

the overall system by, for example, revealing keys. The backend and secure element are tamper resistant while the mobile device only contains encrypted data.

(T9) If a customer is also choosing to maintain the backend and therefore be responsible for all keys, it is possible that key material is unintentionally published or lost in a security breach.
Potentially threatened assets: **(A2), (A3)**

(R9) If a customer chooses to maintain the key material, no countermeasure can be taken in order to mitigate the threat of losing key material.

(E4) Configuration Changer / (E5) Adversary with physical access [7]

(T10) The person responsible for deploying configurations might try to deploy arbitrary configuration parameters and thus, harm the device. Faulty configurations can, for example, be generated by producing manipulated QR codes which contain configurations.
Potentially threatened assets: **(A1), (A3)**

(C10) Authenticated encryption is used to prevent the deployment of untrusted configurations.

(T11) The person responsible for deploying configurations might try to manipulate the configuration that is present on the mobile device. Manipulation includes trying to change certain parameters or to try and change random bits of the configuration.
Potentially threatened assets: **(A1), (A3)**

(C11) Authenticated encryption is used to prevent the deployment of potentially manipulated configurations.

(T12) The person responsible for deploying configurations might try to (intentionally or unintentionally) deploy outdated configurations.
Potentially threatened assets: **(A3)**

(C12) By integrating expiry dates and timestamps as well as version numbers into the configurations, the threat of overwriting configurations with older versions is mitigated.

[7] As the person who changes configurations might be a malicious user, the threats are combined with those of an adversary with physical access as most threats are overlapping.

(T13) The mobile device which is used might be lost by the configuration changing person. It could also be stolen.
Potentially threatened assets: **(A1)**

(C13) As a countermeasure, the mobile device's authentication mechanisms must be used. Also, by only transferring encrypted configurations to the mobile device, no confidential data can be read. As also MACs are used, only valid configurations can be deployed using a lost or stolen mobile device anyhow.

(T14) The person responsible for deploying configurations might try to manipulate the SECURECONFIG application or the mobile device's operating system to try and get access to configurations or to deploy faulty configurations.
Potentially threatened assets: **(A1), (A3)**

(C14) To prevent application modification, the operating system's security mechanisms must be used. No bypassing such as rooted Android operating systems must be allowed. To fully guarantee a non-manipulated application and operating system, mobile devices which use hardware security mechanisms such as a secure element must be used.

(T15) As persons who change configurations have physical access to the secure element, they might try to manipulate the secure element through physical attacks.
Potentially threatened assets: **(A1), (A2), (A3)**

(C15) The secure element must be tamper resistant to avoid the manipulation of processed data such as intermediate keys or a final plain- or ciphertext.

(T16) Denial of service regarding the device which should be updated is possible by either destroying, not updating or repeatedly trying to update the device is possible.
Potentially threatened assets: **(A3)**

(R16) There is no possibility to mitigate all possible forms of a denial of service attack.

(E5) Adversary without physical access

(T17) Denial of service regarding the NFC-based configuration interface is also possible without having physical access to the hardware.
Potentially threatened assets: **(A3)**

(R17) As mentioned in Section 2.1.3, a denial of service attack on NFC communication is possible. However, as the communication range of NFC devices is limited, no countermeasure needs to be taken.

(T18) Without having physical access, sniffing of data transferred using NFC might be possible, although the communication range is very limited.
Potentially threatened assets: **(A1)**

(C18) As a countermeasure, the transferred data is secured using authenticated encryption. Also, the rather short communication range of NFC limits the risk of this threat.

(T19) A man-in-the-middle attack, although infeasible, is still possible on NFC communication. Because of the wireless nature, no physical access is needed.
Potentially threatened assets: **(A1), (A3)**

(C19) By using authenticated encryption, the threat of man-in-the-middle attacks is mitigated. The short communication range of NFC also makes this type of attack highly unlikely, when compared to other wireless techniques.

(T20) Because configurations are downloaded from a backend, an adversary might try to manipulate the download of the content.
Potentially threatened assets: **(A1), (A3)**

(C20) By using HTTPS for downloading configurations from the backend, the possibility of manipulating the downloaded content is erased.

(T21) As the backend might be accessible from the Internet, adversaries could try to directly attack the backend to steal key material or confidential configuration parameters.
Potentially threatened assets: **(A1), (A2), (A3)**

(C21) As mentioned when listing the entities, the backend is assumed to be secure against any kind of attack as securing a web application is not the focus of this master's thesis.

The result of the conducted security and risk analysis shows that countermeasures are taken regarding all threats where mitigating the threat through a countermeasure was possible in the context of this master's thesis. The residual risks that remain are mainly threats which arise through misconfiguration of the provided security features or denial of service attacks. Although it is also possible to mitigate those threats, no countermeasures were taken in the scope of this master's thesis.

5.2 Evaluation of Packet Sizes

To highlight the effect of the implemented security measures regarding transferred packet size, an evaluation was done. For this assessment, the algorithms chosen where *AES-128* and *HMAC-SHA256* which gave the following overheads according to Section 3.3.2 and Section 3.5:

- The overhead that is added to the plaintext before encryption is applied is 74 Byte. The transferred nonce needs 64 Bytes while the other fields such as version or validity account for 10 Bytes.
- The overhead that is applied after encryption for fields such as real-time or cipher spec is 6 Byte plus the 32 Bytes hash that is produced by the chosen *HMAC-SHA256* algorithm.
- Because AES is a block cipher, the plaintext is encrypted in chunks of 16 Bytes (for AES-128). If the plaintext length in Bytes is no multiple of 16, the plaintext is padded which additionally increases the packet size.

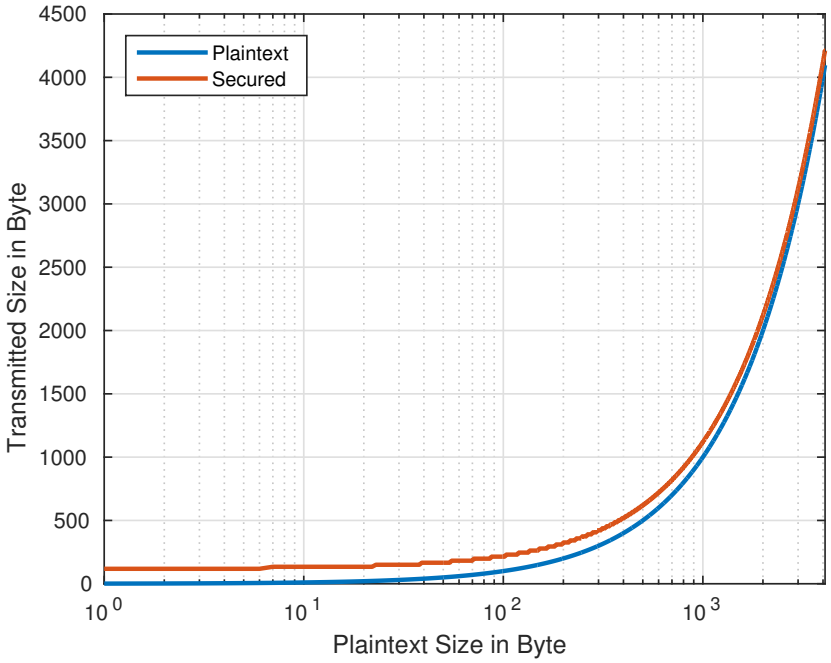


Figure 5.2: Comparison of necessary packet size for unsecured plaintext and secured protocol in the range [1, 4096] Bytes.

Figure 5.2 shows the packet size in bytes that need to be transferred for the unsecured transmission of plaintext and the secured method which is presented in this master's

thesis. The considered range for the plaintext size is 1 byte up to 4096 bytes as 4 kB is the maximum size of QR codes as well as the maximum payload size for standard NDEF messages. As mentioned before, the overhead comprises a static part which is induced by information such as the nonce, real-time or version number that are added and a variable part caused by AES being a block cipher. Figure 5.3 highlights this stepwise increase in overhead as only payload sizes in the range from 1 byte up to 512 bytes are considered for this plot. One aspect which needs to be noted is that because of the overhead, introduced by the implemented security mechanisms, a maximum plaintext size of 3974 bytes can be transferred using NDEF messages and QR codes as a larger plaintext size would lead to an overall packet size greater than 4 kB.

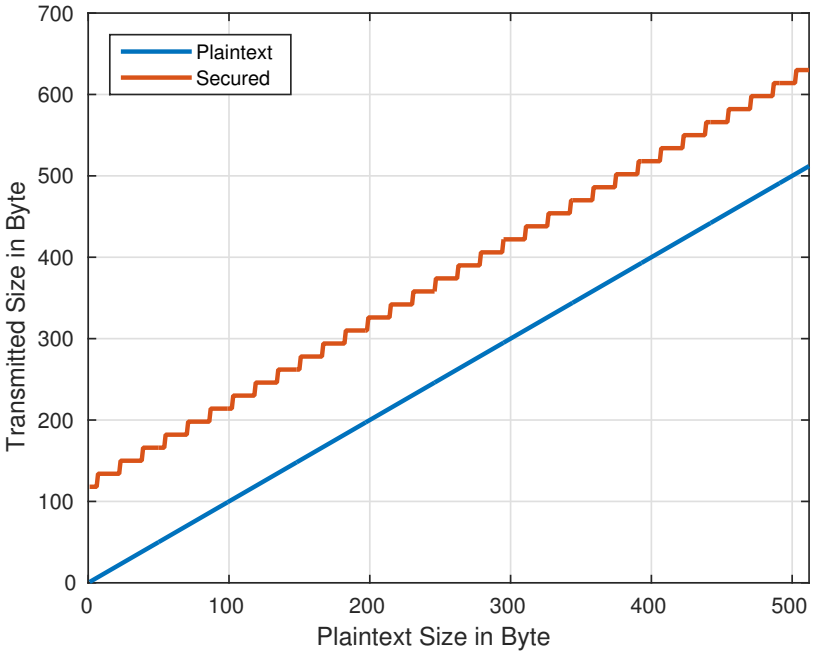


Figure 5.3: Comparison of necessary packet size for unsecured plaintext and secured protocol in the range [1, 512] Bytes, highlighting the blockmode of AES.

The overhead in percentage of the overall packet size is depicted in Figure 5.4. The overhead is calculated according to Equation (5.1). As can be seen there, the overhead is not negligible as for plaintext sizes of 1024 bytes the overhead would still account for 10% of the overall packet size.

$$\text{overhead in } \% = \frac{\text{overhead in bytes}}{\text{plaintext size in bytes + overhead in bytes}} \tag{5.1}$$

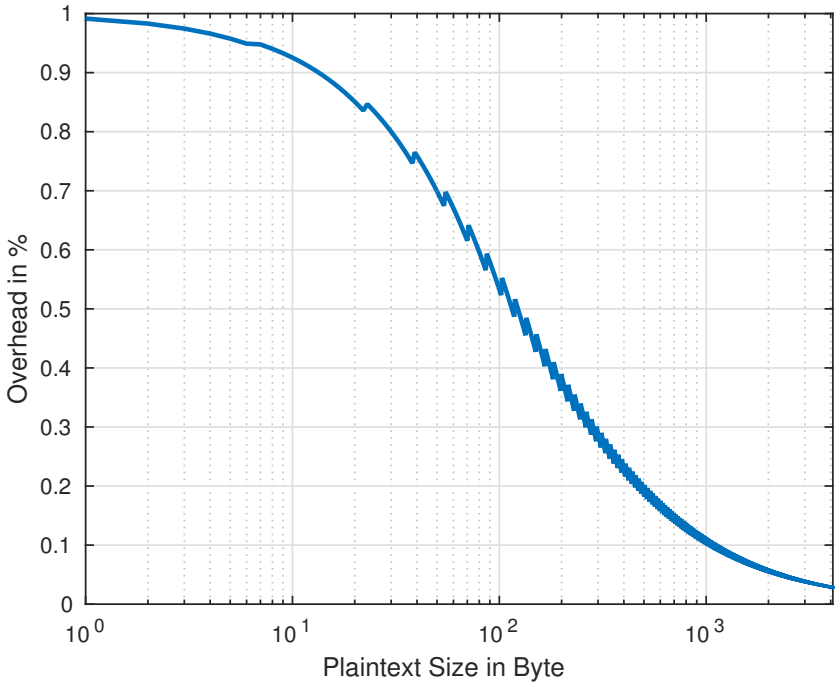


Figure 5.4: Overhead of securing the payload in percent relative to the unsecured transport of plaintext data.

6 Conclusion and Future Work

This master's thesis is concluded with this chapter. First, a conclusion which summarizes the contents of this thesis is given. Also, currently existing limitations of the presented prototype and concept are given. The chapter ends with an outlook regarding potential future work which aims at mitigating some limitations as well as enhancing the current solution.

6.1 Conclusion

In the scope of this master's thesis, an NFC-based configuration interface for the Internet of Things (IoT) was presented. As the concept is based on a hardware concept that was implemented in another thesis, the so-called *mediator* is presented first. Based on this hardware platform and the context of the IoSense project, several possible application scenarios were discussed in this work, highlighting potential use cases of the proposed NFC-based configuration interface. Before the concept was discussed, also all necessary prerequisites were reviewed, and related work was discussed. As was shown there, at the time this master's thesis was concluded, no similar solution to the one proposed in this thesis existed.

After the introductory chapters of this work, the design of the NFC-based configuration interface was discussed. There, the applied security features are highlighted, and reasons for the chosen techniques are given. After that, the implementation of the prototype developed in the context of this master's thesis is illustrated. For each component of the overall system, implementations of the main parts are discussed. Also, the implemented user interface is presented. To evaluate the presented protocol and prototype, a thorough

threat and risk analysis was conducted. Also, the overhead imposed by the applied security mechanisms is analyzed.

Summarizing this thesis, an NFC-based configuration interface for the Internet of Things (IoT) was presented in this thesis. The proposed concept considers security relevant aspects to ensure the confidentiality and integrity of the transferred configuration parameters. Thus, also confidential information such as production relevant settings or key material can be transferred using this concept. The presented mobile device application ensures a good usability of the developed application to reduce possible human errors when using the concept presented in this master's thesis. Also, the third goal defined in Section 1.3, portability, is accomplished as so-called components of the shelf can be used for the backend and mobile device components. By conduction a threat and risk analysis, applied countermeasures for all discovered threats are given. Thus, the security of this system is analyzed and shown. As the evaluation regarding the overhead imposed by those security measures shows, a nearly negligible percentage of about 10% is imposed for data packages of 1 kB by the implemented security mechanisms. As confidential data needs to be transported and also when considering the transfer rates of NFC, this overhead does not influence the applicability of the concept presented in this master's thesis.

6.1.1 Limitations

Although the presented concept aims at improving the current state of the art (Section 2.3.4, some limitations in the presented concept exist. Most of those limitations arise from time limitations of a master's thesis and, therefore, can be easily fixed by future work which is discussed in Section 6.2. The nature of limitations discussed in this section ranges from conceptual decisions up to technical drawbacks of the implemented prototype.

1. As discussed in Section 5.2, the introduction of security mechanisms induces a certain threshold. The evaluation reveals that for a plaintext size of 1 kB, the overhead accounts for roughly 10% of the overall transferred data. The overhead is caused by the inclusion of several security related fields such as a nonce or machine ID as well as by using authenticated encryption which requires transmitting an additional MAC.

2. Based on the core system architecture and the communication protocols introduced in Chapter 3, the mobile device used for device configuration needs to be equipped with a digital camera and an NFC interface. The NFC interface is needed to configure devices. However, the additional need for a camera can be seen as a potential drawback as no simple NFC reader/write can be used as a mobile device in this setting.

3. By using QR and NFC technology as described in Section 3.2 the maximum configuration package size is restricted to a size of around 4 kB. The implementation

of the security features mentioned in Section 3.5 further decreases the effective maximum plaintext size to 3974 bytes as discussed in Section 5.2.

4. The application deployed on the mobile device was developed for the Android operating system because of simplicity reasons. However, using Android restricts the range of mobile devices to such devices that can run the Android operating system.

5. The secure elements software components were developed on a specific Infineon secure element, thus restricting the applicability of those software components to exactly that specific hardware element.

6. The backend is implemented in JavaScript using nodeJS functionality and libraries as discussed in Section 4.1.2. By using those technologies, suitable hardware to operate the backend on is restricted to those capable of running the necessary software.

7. As shown in Section 3.3.2, timestamps are needed to verify the validity of configuration packages. However, for Android systems, there is no way to ensure that the system time was not modified. The only way is to use the so-called *elapsed real-time* which measures the time since the device was booted including sleep phases. It is not possible to manipulate that counter, however rebooting the device renders all previously created timestamps as useless. For this reason, when rebooting the mobile device, all stored configurations are lost.

8. Because private-key encryption is used, a private key which is used for decryption needs to be present at the secure element before configuring it for the first time. The private key can be stored in a secure storage, however, by requiring the key to being present before applying the first configuration may require for the device manufacturer to deploy those keys.

9. Although authenticated encryption is used to secure the transferred configuration parameters, no direct authentication of the person and mobile device applying the configuration is required. Thus, by knowing the private key and the structure used to represent configuration parameters, an attacker can apply arbitrary configuration settings.

10. To assemble configurations, a backend software is needed as discussed in Section 3.2. However, especially for smart home scenarios, users might not want to use an additional component when composing and applying configurations.

6.2 Future work

A couple of limitations listed in Section 6.1.1 could be solved by doing further work in the context of NFC-based configuration. Also, enhancements to the current project state might be possible. This section lists both, future work with a focus on improving the limitations listed in Section 6.1.1 as well as possible enhancements.

1. An evaluation concerning energy efficiency needs to be conducted. To do so, energy measurements at instruction level need to be collected to be able to evaluate the currently implemented prototype concerning energy consumption.

2. A mechanism for password-based key exchange that was implemented in a second master's thesis done in the context of the *IoSense* project could be integrated into the proposed NFC-based configuration solution. By incorporating the so-called *SPAKE* [AP05] algorithm, the problem of unauthenticated configuration as well as the issue of initial key distribution could be mitigated.

3. In addition to the QR- and NFC-based approach as presented in Section 3.2 an approach that only uses NFC could be developed. By doing so, the drawback of needing a camera in the mobile device when transferring configurations is eliminated.

4. Currently, only configurations that are composed by hand can be applied to devices. However, it might be useful to be able to copy a device's configuration, for instance in the case of a broken down manufacturing device. The copied configuration could then directly be applied to the new machine replacing the broken one.

5. As mentioned in Section 6.1.1, a dedicated backend is needed to compose configuration packages. However, especially in a smart home scenario, creating and maintaining configurations directly in the mobile device's application might be very useful. Therefore, an application that enables end-users of changing existing configurations could be developed.

6. For the prototype implemented within the scope of this master's thesis, an Android smartphone was chosen as a mobile device. To not be restricted by that choice, a special mobile device that is fulfilling all requirements stated in Section 4.1.1 could be developed. Such a device could be equipped with several security relevant features that are present in each Android device, such as a non-changeable real-time clock and an included secure element.

Bibliography

[ABCS06] Ross Anderson, Mike Bond, Jolyon Clulow, and Sergei Skorobogatov. Cryptographic processors-a survey. *Proceedings of the IEEE*, 94(2):357–369, 2006.

[ACM13] Alessandro Armando, Gabriele Costa, and Alessio Merlo. Bring your own device, securely. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1852–1858. ACM, 2013.

[ADF07] Manfred Aigner, Sandra Dominikus, and Martin Feldhofer. A system of secure virtual coupons using NFC technology. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pages 362–366. IEEE, 2007.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In *Cryptographers' Track at the RSA Conference*, pages 191–208. Springer, 2005.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 531–545. Springer, 2000.

[Bro01] David L Brock. The electronic product code (epc). *Auto-ID Center White Paper MIT-AUTOID-WH-002*, 2001.

[BSDF12] Balazs Benyo, Balint Sodor, Tibor Doktor, and Gergely Fördős. Student attendance monitoring at the university using NFC. In *Wireless Telecommunications Symposium (WTS), 2012*, pages 1–5. IEEE, 2012.

[CW08] Nicolaj Bjerregaard Christensen and Stefan Wagner. Using near field communication technology to achieve near-zero-configuration of sensors. In *Proceedings of the 5th Annual International Conference on Mobile and*

Ubiquitous Systems: Computing, Networking, and Services, page 38. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[CYHI+03] Diane J Cook, G Michael Youngblood, Edwin O Heierman III, Karthik Gopalratnam, Sira Rao, Andrey Litvin, and Farhan Khawaja. MavHome: An Agent-Based Smart Home. In *PerCom*, volume 3, pages 521–524, 2003.

[DHM06] Jing Deng, Richard Han, and Shivakant Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 292–300. ACM, 2006.

[DM08] Mohsen Darianian and Martin Peter Michael. Smart home mobile RFID-based Internet-of-Things systems and services. In *2008 International conference on advanced computer theory and engineering*, pages 116–120. IEEE, 2008.

[DMK+13] Norbert Druml, Manuel Menghin, Daniel Kroisleitner, Christian Steger, Reinhold Weiss, Armin Krieg, Holger Bock, and Josef Haid. Emulation-Based Fault Effect Analysis for Resource Constrained, Secure, and Dependable Systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 337–344. IEEE, 2013.

[DMK+14] Norbert Druml, Manuel Menghin, Adnan Kuleta, Christian Steger, Reinhold Weiss, Holger Bock, and Josef Haid. A flexible and lightweight ECC-based authentication solution for resource constrained systems. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 372–378. IEEE, 2014.

[DSTW12] Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Sandeep Tamrakar, and Christian Wachsmann. SmartTokens: Delegable access control with NFC-enabled smartphones. In *International Conference on Trust and Trustworthy Computing*, pages 219–238. Springer, 2012.

[DXHL14] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[EF15] Claudia Eckert and Niels Fallenbeck. Industrie 4.0 meets IT-Sicherheit: eine Herausforderung! *Informatik-Spektrum*, 38(3):217–223, 2015.

[EOM+09] William Enck, Machigar Ongtang, Patrick Drew McDaniel, et al. Understanding Android Security. *IEEE security & privacy*, 7(1):50–57, 2009.

[EOMC11] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011.

[fSEC+04] International Organization for Standardization/International Electrotechnical Commission et al. ISO/IEC 18092 Information technology—Telecommunications and information exchange between systems—Near Field Communication—Interface and Protocol (NFCIP-1). *ISO/IEC*, 18092, 2004.

[fSEC+05] International Organization for Standardization/International Electrotechnical Commission et al. ISO/IEC 21481 Information technology—Telecommunications and information exchange between systems—Near Field Communication—Interface and Protocol (NFCIP-2). *ISO/IEC*, 21481, 2005.

[FT11] Luka Finžgar and Mira Trebar. Use of NFC and QR code identification in an electronic ticket system for public transport. In *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*, pages 1–6. IEEE, 2011.

[GSMM09] Stefano Levialdi Ghìron, Serena Sposato, Carlo Maria Medaglia, and Alice Moroni. NFC ticketing: A prototype and usability test of an NFC-based virtual ticketing application. In *Near Field Communication, 2009. NFC'09. First International Workshop on*, pages 45–50. IEEE, 2009.

[Har06] Richard Harper. *Inside the smart home*. Springer Science & Business Media, 2006.

[Hat03] Lynn Hathaway. National policy on the use of the advanced encryption standard (AES) to protect national security systems and national security information. *National Security Agency*, 23, 2003.

[HB06] Ernst Haselsteiner and Klemens Breitfuß. Security in near field communication (NFC). In *Workshop on RFID security*, pages 12–14, 2006.

[HDK+14] Andrea Höller, Norbert Druml, Christian Kreiner, Christian Steger, and Tomaz Felicijan. Hardware/software co-design of elliptic-curve cryptography for resource-constrained applications. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[HL10] Dae-Man Han and Jae-Hyun Lim. Smart home energy management system using IEEE 802.15. 4 and zigbee. *IEEE Transactions on Consumer Electronics*, 56(3):1403–1410, 2010.

[HMEK16] Jan Haase, Dominik Meyer, Marcel Eckert, and Bernd Klauer. Wireless sensor/actuator device configuration by NFC. In *2016 IEEE International Conference on Industrial Technology (ICIT)*, pages 1336–1340. IEEE, 2016.

[Jaz14] Nasser Jazdi. Cyber physical systems in the context of Industry 4.0. In *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*, pages 1–4. IEEE, 2014.

[JVW+14] Qi Jing, Athanasios V Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu. Security of the internet of things: Perspectives and challenges. *Wireless Networks*, 20(8):2481–2501, 2014.

[K+03] Patrick Kinney et al. Zigbee technology: Wireless control that simply works. In *Communications design conference*, volume 2, pages 1–7, 2003.

[KCB97] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. HMAC: Keyed-hashing for message authentication. 1997.

[KHJ+99] Yoram Koren, Uwe Heisel, Francesco Jovane, Toshimichi Moriwaki, Gumter Pritschow, Galip Ulsoy, and Hendrik Van Brussel. Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology*, 48(2):527–540, 1999.

[KHW98] Yoram Koren, S Jack Hu, and Thomas W Weber. Impact of manufacturing system configuration on performance. *CIRP Annals-Manufacturing Technology*, 47(1):369–372, 1998.

[KKYL11] Divyan M Konidala, Dae-Young Kim, Chan-Yeob Yeun, and Byoung-Cheon Lee. Security framework for RFID-based applications in smart home environment. *Journal of Information Processing Systems*, 7(1):111–120, 2011.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *Annual International Cryptology Conference*, pages 310–331. Springer, 2001.

[LBK15] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[LdIVJ07] Diego López-de Ipiña, Juan Ignacio Vazquez, and Iker Jamardo. Touch computing: Simplifying human to environment interaction through NFC technology. *1as Jornadas Científicas sobre RFID*, 21, 2007.

[Lee08] Edward A Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE, 2008.

[LH12] Minkyu Lee and Dongsoo Han. QRLoc: User-involved calibration using quick response codes for Wi-Fi based indoor localization. In *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, pages 1460–1465. IEEE, 2012.

[LHH+15] Christian Lesjak, Daniel Hein, Michael Hofmann, Martin Maritsch, Andreas Aldrian, Peter Priller, Thomas Ebner, Thomas Ruprechter, and Günther Pregartner. Securing smart maintenance services: Hardware-security and TLS for MQTT. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1243–1250. IEEE, 2015.

[LHJ08] Antti Lahtela, Marko Hassinen, and Virpi Jylha. RFID and NFC in healthcare: Safety of hospitals medication care. In *2008 Second International Conference on Pervasive Computing Technologies for Healthcare*, pages 241–244. IEEE, 2008.

[LHW15] Christian Lesjak, Daniel Hein, and Johannes Winter. Hardware-security technologies for industrial IoT: TrustZone and security controller. In *Industrial Electronics Society, IECON 2015-41st Annual Conference of the IEEE*, pages 002589–002595. IEEE, 2015.

[LRB+14a] Christian Lesjak, Thomas Ruprechter, Holger Bock, Josef Haid, and Eugen Brenner. ESTADO—Enabling smart services for industrial equipment through a secured, transparent and ad-hoc data transmission online. In *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for*, pages 171–177. IEEE, 2014.

[LRB+14b] Christian Lesjak, Thomas Ruprechter, Holger Bock, Josef Haid, and Eugen Brenner. Facilitating a secured status data acquisition from industrial equipment via NFC. *Journal of Internet Technology and Secured Transactions (JITST)*, 2014.

[LRH+14] Christian Lesjak, Thomas Ruprechter, Josef Haid, Holger Bock, and Eugen Brenner. A secure hardware module and system concept for local and remote industrial embedded system identification. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–7. IEEE, 2014.

[LT02] Chih-Chung Lu and Shau-Yin Tseng. Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter. In *Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on*, pages 277–285. IEEE, 2002.

[MF10] Friedemann Mattern and Christian Floerkemeier. From the Internet of Computers to the Internet of Things. In *From active data management to event-based systems and more*, pages 242–259. Springer, 2010.

[MFMP07] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. A common criteria based security requirements engineering process for the development of secure information systems. *Computer standards & interfaces*, 29(2):244–253, 2007.

[MLY05] Suvda Myagmar, Adam J Lee, and William Yurcik. Threat modeling as a basis for security requirements. In *Symposium on requirements engineering for information security (SREIS)*, volume 2005, pages 1–8, 2005.

[MRT12] Alfredo Matos, Daniel Romao, and Paulo Trezentos. Secure hotspot authentication through a near field communication side-channel. In *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 807–814. IEEE, 2012.

[MSDPC12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[MVH12] Keith W Miller, Jeffrey M Voas, and George F Hurlburt. BYOD: Security and Privacy Considerations. *It Professional*, 14(5):53–55, 2012.

[NFC11a] NFC in Public Transport. Technical report, NFC Forum, January 2011.

[NFC11b] Type 4 tag operation specification. Technical Report T4TOP 2.0, NFC Forum, 06 2011.

[NLY13] Huansheng Ning, Hong Liu, and Laurence T Yang. Cyberentity security in the Internet of Things. *Computer*, (4):46–53, 2013.

[PFT+14] Sandor Plosz, Arsham Farshad, Markus Tauber, Christian Lesjak, Thomas Ruprechter, and Nuno Pereira. Security vulnerabilities and risks in industrial usage of wireless communication. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014.

[PRR08] Marc Pasquet, Joan Reynaud, and Christophe Rosenberger. Secure payment with NFC mobile phone in the SmartTouch project. In *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on*, pages 121–126. IEEE, 2008.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[RLG+15] Michael Rüßmann, Markus Lorenz, Philipp Gerbert, Manuela Waldner, Jan Justus, Pascal Engel, and Michael Harnisch. Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries. *Boston Consulting Group*, 2015.

[RNL11] Rodrigo Roman, Pablo Najera, and Javier Lopez. Securing the internet of things. *Computer*, 44(9):51–58, 2011.

[RRC04] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. Tamper resistance mechanisms for secure embedded systems. In *VLSI Design, 2004. Proceedings. 17th International Conference on*, pages 605–611. IEEE, 2004.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International Workshop on Fast Software Encryption*, pages 371–388. Springer, 2004.

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation. In *IFIP International Information Security Conference*, pages 557–570. Springer, 2015.

[SFG09] Guenther Starnberger, Lorenz Froihofer, and Karl M Goeschka. QR-TAN: Secure mobile transaction authentication. In *Availability, Reliability and Security, 2009. ARES'09. International Conference on*, pages 578–583. IEEE, 2009.

[SOM14] Fadi Shrouf, Joaquin Ordieres, and Giovanni Miragliotta. Smart factories in industry 4.0: a review of the concept and of energy management approached in production based on the Internet of things paradigm. In *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 697–701. IEEE, 2014.

[Soo08] Tan Jin Soon. QR code. *Synthesis Journal*, 2008:59–78, 2008.

[SPS+10] Rainer Steffen, Jorg Preissinger, Tobias Schöllermann, Armin Müller, and Ingo Schnabel. Near field communication (NFC) in an automotive environment. In *International Workshop on Near Field Communication*, pages 15–20, 2010.

[SS04] Frank Swiderski and Window Snyder. *Threat modeling*. Microsoft Press, 2004.

[Sta14] John A Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014.

[Ste66] Karl Steinbuch. *Die informierte Gesellschaft*. Deutsche Verlags-Anstalt, 1966.

[SU15] Sadia Syed and M Ussenaiah. The rise of Bring Your Own Encryption (BYOE) for secure data storage in Cloud databases. In *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on*, pages 1463–1468. IEEE, 2015.

[SWW15] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and privacy challenges in industrial internet of things. In *Proceedings of the 52nd Annual Design Automation Conference*, page 54. ACM, 2015.

[Tho12] Gordon Thomson. BYOD: enabling the chaos. *Network Security*, 2012(2):5–8, 2012.

[VDWP09] Gauthier Van Damme, Karel Wouters, and B Preneel. Practical experiences with NFC security on mobile phones. *Proceedings of the RFIDSec*, 9:27, 2009.

[VOZ+12] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M McCune. Trustworthy Execution on Mobile Devices: What security properties can my mobile platform give me? In *International Conference on Trust and Trustworthy Computing*, pages 159–178. Springer, 2012.

[Wan11] Roy Want. Near field communication. *IEEE Pervasive Computing*, 3(10):4–7, 2011.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[Zha15] Hongwen Zhang. Bring your own encryption: balancing security with practicality. *Network Security*, 2015(1):18–20, 2015.