Secure RFID Applications for the Internet of Things

by

Sandra Dominikus

A PhD Thesis
Submitted to the Faculty of Computer Science in Partial Fulfillment of the Requirements for the PhD Degree

Assessors

Karl Christian Posch (Graz University of Technology, Austria)
Rafael Pous Andrés (Universitat Pompeu Fabra, Spain)

December 2011

Institute for Applied Information Processing and Communications (IAIK)
Faculty of Computer Science
Graz University of Technology, Austria

Preface

This thesis documents the essence of my research done over the past years. I had to interrupt work several times for family reasons, that is to say, to give birth to and take care of my four children. Every time I resumed work, the state of technology had changed and new insights had to be taken into account. However, considerable benefits have been drawn from this rather unusual academic career, as I had to deal with an even wider range of various aspects of the topic. The thesis subsumes the most important findings of my latest research.

Sandra Dominikus
Graz, December 2011

Abstract

The Internet of the future, the so-called *Internet of Things*, will be a huge network of heterogeneous smart devices. This network is on the way to revolutionize the mobile world of the present. Pervasive use of mobile devices, some of them with computational power we are unaware of, has already become part of our lives. Most of these devices are able to communicate with other mobile items. In the future, these communication capabilities will be even more extended. Smart items will be linked together by a large network in order to provide new applications. The *Internet of Things* will have different requirements than the conventional Internet, because the computational capacity of the participants on the network range from powerful to very limited. This thesis focuses on one of the most constrained items within the *Internet of Things*: Passive RFID tags. We deal with three essential topics when developing secure RFID applications for the Internet of Things: Security-enhanced RFID tags, building applications based on these tags, and their connection to the Internet.

As the capabilities of RFID tags grow constantly, new complex applications will be possible. Some of these applications will require the use of security measures. The constraints for hardware in passive RFID tags are very rigid in terms of power, timing, size, and memory. Therefore, not all security mechanisms are possible for the use in these tags. In this work we investigate on cryptographic primitives and security protocols that are able to provide strong security for passive RFID tags.

When developing security protocols for RFID tags, simulation and prototyping are essential steps in the design flow. First implementations could fail or perform inadequately. Software simulations give an early indication whether the protocol works and whether timing constraints can be met. However, a successful simulation cannot guarantee that the application will work on dedicated hardware. Prototyping on a hardware platform is thus also essential. In this thesis we present a design flow for secure RFID applications and software tools to speed up the simulation and prototyping processes. The faster the design phase, the shorter the time to market.

The range of possible RFID applications will be enlarged by remote access to RFID tags. In order to provide online communication with RFID tags, we describe a concept on how to connect RFID tags to the Internet. As for all networks, secure communication is an indispensable feature also for the *Internet of Things*. Therefore, we extend the proposed communication concept by a security layer based on IPSec mechanisms. This security layer provides secure end-to-end communication with RFID tags on the Internet.

Acknowledgements

This thesis was made possible by the kind support and help of several people. I want to thank the persons that supported me during my work: I am thankful for the constant advice and helpful feedback which I received from my supervisor Karl-Christian Posch. I want to thank my second assessor Rafael Pous for taking the time for reading this thesis and traveling to Graz. I also want to thank my present and former colleagues for many fruitful discussions and for pleasant company during the coffee breaks. Special thanks go to Michael Hutter, Mario Kirschbaum, Thomas Plos, and Jörn-Marc Schmidt who gave me valuable feedback on this thesis. I would like to mention Manfred Aigner and Hannes Wolkerstorfer at this point for being good friends in professional as well as private matters. Furthermore, I would like to thank Lieselotte Hölbling for the sound review of the text.

I am thankful for the help from my family. Without it, writing of the thesis would not have been possible. I am greatly indebted to my husband Heinz for his patience, encouragement and constant support. I also want to thank my parents, Anni and Hermann, for being there for me throughout all my life. I thank my parents in law, Irene and Hermann, for always being willing and ready to help with family matters. Special thanks go to my children, Manuel, Nikolas, Benjamin, and Florian, for bringing me down to earth and for always reminding me of the really important things in life.

Sandra Dominikus
Graz, December 2011

Table of Contents

List of Publications

1. Sandra Dominikus. Multi-Purpose Hash Module: VLSI Desing of MD4-Family Hash Algorithms. Master's thesis, Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria, January 2002.

2. Sandra Dominikus. A Hardware Implementation of MD4-Family Hash Algorithms. In 9th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2002. Dubrovnik, Croatia, 15-18 September, 2002, Proceedings, pages 1143 -1146, IEEE.

3. Sandra Dominikus and Stefan Mangard. Eine universelle AES Hardware Architektur. In Peter Söser and Andreas Buslehner, editors, *Austrochip 2002, Procedings*. October 2002, Graz, Austria, pages 65–72. ISBN 3-9501635-0-6.

4. Stefan Mangard, Manfred Aigner, and Sandra Dominikus. A Highly Regular and Scalable AES Hardware Architecture. In IEEE Transactions on Computers, April 2003, volume 52, pages 438 - 491.

5. Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong Authentication for RFID Systems using the AES Algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 2004, Proceedings*, Lecture Notes in Computer Science, volume 3156, pages 357 - 370. Springer.

6. Martin Feldhofer, Manfred Aigner, and Sandra Dominikus. An Application of RFID Tags using Secure Symmetric Authentication. In Panagiotis Georgiadis, Stefanos Gritzalis, and Giannis F. Marias, editors *1st International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing – SecPerU 2005, Santorini Island, Greece, July 2005, Proceedings*, pages 43 - 49, Diavlos Publications.

7. Norbert Pramstaller, Stefan Mangard, Sandra Dominikus, and Johannes Wolkerstorfer. Efficient AES Implementations on ASICs and FPGAs. In H. Dobbertin, V. Rijmen, and A. Sowa, editors *Fourth Workshop on the Advanced Encryption Standard, AES4 - State of the Crypto Analysis', Bonn, Germany, May 10-12, 2005, Proceedings*, Lecture Notes in Computer Science, volume 3373, pages 98 - 112. Springer.

8. Sandra Dominikus, Maria Elisabeth Oswald, and Martin Feldhofer. Symmetric Authentication for RFID Systems in Practice. In Proceedings of Workshop on RFID and Light-Weight Crypto, July 2005, pages 25 - 31.

9. Sandra Dominikus, Maria Elisabeth Oswald, and Martin Feldhofer. Practical Security for RFID: Strong Authentication Protocols. In Patrick Horster, editor *D.A.CH Mobility 2006, Proceedings*, pages 187 - 200, Syssec. ISBN 3-00-019635-8.

10. Sandra Dominikus and Manfred Aigner. mCoupons: An Application for Near Field Communication (NFC). In *IEEE International Symposium on Ubisafe Computing (UbiSafe-07) Niagara Falls, Ontario, Canada, May 2007, Proceedings*, pages 421–428, IEEE.

11. Manfred Aigner, Sandra Dominikus, and Martin Feldhofer. A System of Secure Virtual Coupons Using NFC Technology. In *Workshop on Pervasive RFID/NFC Technology and Applications (PerTec07), New York, USA, March 2007, Proceedings*, pages 362–366, IEEE Computer Society Press.

12. Sandra Dominikus, Manfred Aigner, and Stefan Kraxberger. Passive RFID Technology for the Internet of Things. In *International Conference for Internet Technology and Secured Transactions 2010, ICITST 2010, London, UK, November 2010, Proceedings*, pages 1–8, IEEE.

13. Sandra Dominikus, Stefan Kraxberger, Manfred Aigner, and Hannes Gross. Low-cost RFID Tags as IPv6 Nodes in the Internet of Things. In Tieyan Li, Chao-Hsien Chu, Ping Wang and Guilin Wang, editors *Radio Frequency Identification System Security, RFIDsec'11 Asia Workshop Proceedings*, Cryptology and Information Security Series, volume 6, pages 114–128, April 2011, IOSPress. ISBN 978-1-60750-721-5.

14. Sandra Dominikus. MedAssist - A Privacy Preserving Application using RFID Tags. In *IEEE International Conference on RFID-Technology and Applications, RFID-TA 2011, September 2011, Barcelona, Spain, Proceedings.*

15. Sandra Dominikus, and Stefan Kraxberger. Secure Communication with RFID tags in the Internet of Things. In *Journal of Security and Communication Networks, Special Issue on Protecting the Internet of Things*, John Wiley Sons Ltd, 2011.

List of Tables

List of Figures

Acronyms

AES	Advanced Encryption Standard
AFI	Application Family Identifier
AH	Authentication Header
CA	Certification Authority
CoA	Care-of Address
CN	Correspondent Node
CRC	Cyclic Redundancy Check
DAA	Direct Anonymous Attestation
DH	Diffie-Hellman
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EIRP	Equivalent Isotropically Radiated Power
EOF	End of Frame
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Service
ERP	Effective Radiated Power
ESP	Encapsulating Security Payload
EUI	Extended Unique Identifier
GID	General Identifier
GUI	Graphical User Interface
HA	Home Agent
HF	High Frequency
HMAC	Hash-based Message Authentication Code
IAIK	Institute for Applied Information Processing and Communications
ICMFG	Integrated Circuit Manufacturer Code
ICV	Integrity Check Value
ID	Identifier
IKE	Internet Key Exchange
IKEv2	Internet Key Exchange version 2
IoT	Internet of Things
IPSec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISO	International Organization for Standardization
LAN	Local Area Network

MIPv6	Mobile Internet Protocol version 6
MAC	Message Authentication Code
MAC	Media Access Control
NIST	National Institute of Standards and Technology
NFC	Near Field Communication
PET	Privacy-enhancing Technology
PETRA	Protocol Evaluation Tool for RFID Applications
PKI	Public Key Infrastructure
POS	Point of Sales
PRNG	Pseudo Random-Number Generator
ProtEx	Prototyping Express
RF	Radio Frequency
RFC	Request for Comments
RFID	Radio Frequency Identificaton
SA	Security Association
SADB	Security Association Database
SHA	Secure Hash Algorithm
SOF	Start of Frame
SPD	Security Policy Database
SPI	Security Parameter Index
TCP	Transmission Control Protocol
TPM	Trusted Platform Module
UDP	User Datagram Protocol
UHF	Ultra High Frequency
UID	Unique Identifier

1 Introduction

Nowadays we are surrounded by mobile devices with computational power that human beings often do not realize. These devices communicate even without human interaction. The exchange of data and the capability of these items to act smart, up to a certain degree, make it possible to develop numerous new and useful applications. The vision of the *Internet of Things* is a network, where all types of smart devices are interconnected the same way as nodes within the conventional Internet. This network will consist of heterogeneous devices, some of them computationally powerful, some very restricted. This thesis concentrates on some of the least powerful devices within the *Internet of Things*: Passive RFID tags.

As the RFID topic became popular, research on advanced functionality of RFID tags soon became a major issue. Although the computational power of RFID tags is relatively low due to the rigid power and timing constraints, these tags are already able to do complex computations, store a certain amount of data, or can be provided with sensors. Advanced functionality of RFID tags gives way to innovative applications. Many of these applications will require security measures. Due to the rigid constraints in RFID environments, new security primitives and protocols that meet these constraints have to be developed. We investigate on cryptographic hardware modules feasible for RFID systems. Developing security protocols based on these hardware modules, strong security for RFID applications is provided.

In application design, a typical design flow can be identified. We describe six phases starting from the application specification up to the production of tags. Within this design flow, simulation and prototyping are important steps for validation and evaluation of the application specification. These design steps can be sped up significantly by the use of software tools. We discuss the use

of software and prototyping tools for RFID and present two particular tools, namely *PETRA* and *ProtEx*.

In order to profit from enhanced tag functionality, we have explored new communication possibilities for RFID tags. It is predictable that communication with other mobile devices and exchange of data on the Internet will be an important feature for new RFID applications. Communicating on an open network like the Internet, security is an important requirement. Secure communication has been one of the main success factors of the conventional Internet. There are two sides to security. On the one hand, many applications would not have been possible: Security is a prerequisite of e-banking or e-commerce to guarantee confidential and authenticated messaging. On the other hand, confidence of users in the Internet is a critical success factor for this media. The same will hold true for the *Internet of Things*. The difference to the traditional Internet will be the diversity of the participants. Powerful devices like mobile phones or PDAs can easily adopt traditional security protocols. For less powerful items, like RFID tags, new security concepts have still to be developed. Besides security measures for offline RFID applications, we present a security layer for online communication with RFID tags on the Internet.

In the last years we concentrated on the feasibility of cryptographic primitives and security protocols for passive RFID tags and their integration into secure applications. We have developed new security approaches for RFID systems and have designed a concept on the integration of these security-enhanced RFID tags into the *Internet of Things*. The thesis outlines our contribution to the three main topics of secure RFID applications on the *Internet of Things*: Cryptographic primitives for passive RFID tags, design and prototyping of security-enhanced RFID tags and protocols for new applications, and the secure connection of passive RFID tags to the *Internet of Things*.

Cryptographic Primitives

This part of the thesis focuses on cryptographic primitives that can be used in RFID tags. Passive RFID devices have very rigid hardware constraints in terms of power, time and size. Therefore, standard hardware implementations of cryptographic algorithms usually cannot be applied without adaption to these constraints. In the Institute for Applied Information Processing and Communications (IAIK) we have focused on the research on standardized cryptographic primitives, like the Advanced Encryption Standard (AES). Strong security can be provided in an optimal way by standardized and well-tested algorithms. Our research includes the hardware-efficient implementation of the primitives as well as the implementation of communication protocols and applications using these primitives. The AES algorithm has already been optimized for hardware implementation during its design procedure. Therefore, AES was the first choice to prove that hardware implementations of strong security primitives can be feasible for the constraints in passive RFID tags.

Based on the experience we had gained through our prior work on hardware implementation of the AES, the first AES module to meet the rigid constraints

in RFID environments was designed. The result of this development was first published in 2005. Meanwhile, many other symmetric and asymmetric primitives have been designed and implemented for the use in passive RFID tags. In addition, quite a number of proprietary light-weight algorithms, which meet the RFID constraints even better than standardized primitives, have been proposed for this purpose. Many of these algorithms were already broken or showed weaknesses during review. We took this as encouragement to exclusively use standardized algorithms for strong security requirements.

Security Protocols

Based on cryptographic primitives suitable for RFID tags, security protocols for RFID applications can be built. The development of these protocols, for RFID as well as for any other technology, is an iterative process with various phases. Thus, we propose a formal design flow for this process consisting of six phases starting from the first idea to the production of new RFID tags and readers. Our design flow focuses on secure RFID applications, therefore the development of security measures is a separate step in this concept. Simulation and prototyping are important steps during the development of RFID applications, as modifications to the previously designed protocols can be done with relatively small effort during these steps.

We did research on how to ease the task of simulation and prototyping. The outcome of our research are the simulation tool *PETRA*, as well as the simulation and fast-prototyping tool called *ProtEx*. These tools provide basic functionality of existing RFID communication standards. Additional functionality required for the implementation of the new application can be easily programmed by the application developer. The application protocols are simulated and its functionality is verified in software. Hardware prototyping is a more complex task. Existing RFID tags, in most cases, do not support the required functionality to execute new protocols. Therefore, new tags would have to be produced to get hardware prototypes. Typically, this is an expensive and time-consuming process. In the prototyping part of *ProtEx*, we use the IAIK *DemoTag* as a programmable RFID platform that performs like a "real" RFID tag in the field [Dem]. The IAIK *DemoTag* supports different RFID communication standards and can be programmed with additional functionality by the user. The goal of *ProtEx* is to use the same resources for simulation as well as for prototyping and to accelerate the process from the first idea of a protocol to a working hardware prototype.

Connection to the *Internet of Things*

As for the *Internet of Things*, a clear concept that allows the heterogenous participants on this network to communicate still does not exist. In principle, two different approaches can be thought of: The first approach asks for every participant to speak the same "language" (communication protocol, respectively).

The second approach allows the items to communicate using their standard protocol and then use a translator for the standard communication protocol within the *Internet of Things*. It is most likely that the common "language" on the *Internet of Things* will be the Internet Protocol version 6 (IPv6) as this protocol has a huge address space feasible also for billions of potential participants on the *Internet of Things*.

We present a concept which enables passive RFID tags to connect to the Internet via IPv6. The implementation of the entire IPv6 functionality requires a high amount of computational and power resources. That is the reason why our concept is based on the second approach previously mentioned. The tags themselves do not "speak" IPv6 but their messages are translated by the reader. In this way, we shift most of the complexity to the reader, which has more computational power than the tag. As we consider security a crucial part of network communication, we describe a security layer built upon this communication concept. This security layer is based on IPSec mechanisms, which are used on the traditional Internet.

Due to the diversity of the participants some of the paradigms of the traditional Internet do not hold true for the *Internet of Things*. One of the most striking differences is the change from an "always-on" to a "sometimes-on" or even nearly "always-off" technology: The traditional Internet was mainly designed for participants that are statically connected to the network and are almost always online. Many Internet protocols are built upon this assumption. This is not a valid assumption within the *Internet of Things*; many of the participants are mobile and connect only during a fraction of their lifetime to the network. Sensor nodes, for instance, are only online about one percent of their lifetime. Having this in mind, we propose additional features, which could also handle communication with nearly "always-off" devices. We describe concepts on how to handle this scenario in our system.

1.1 Our Contribution

At the beginning of my studies, I started to research on efficient hardware implementations of cryptographic algorithms. Together with Manfred Aigner, Stefan Mangard, Johannes Wolkerstorfer, and Norbert Pramstaller, we succeeded in finding resource-efficient implementations of SHA-1, SHA-256 and AES [Dom02], [DM02], [MAD03], [PMDW05]. As RFID technology became popular, it was common to claim that strong cryptography is far too demanding for the use on passive RFID tags. Based on the experience we had gained on the implementation of strong cryptographic algorithms, we were confident that algorithms like the AES could be also feasible for very rigid design constraints.

In 2004, we designed an AES module in hardware to meet the requirements of passive RFID tags. My colleague Martin Feldhofer developed a low-power implementation of AES, which based on the prior research work we did on this topic. Based on these findings, I implemented the first authentication protocol for RFID systems using the AES algorithm. For this purpose I developed the

first simulation software for an RFID communication standard (namely the ISO-18000-3 standard [Int04b]). This simulation software, called *PETRA* is able to do functional testing of the RFID communication as well as timing simulation of the implemented protocols. This software was used to research on the feasibility and integration of the proposed authentication protocol into an existing RFID communication standard.

Based on the successful hardware implementation of the AES algorithm for RFID tags, we claimed that even more demanding cryptographic algorithms than the AES were feasible for passive RFID technology. At this time, this was not the common opinion. With the help of my colleagues Manfred Aigner, Elisabeth Oswald and Martin Feldhofer we did convincing work in the research community [DOF05], [DOF06]. In order to push the research on secure RFID technology further, we used the AES hardware module to develop several secure RFID applications. The details of this contribution can be found in Chapter 3 as well as in the publications [FDW04] and [FAD05].

A system of virtual mobile coupons (mCoupons) was one of the RFID applications including security protocols. This was a joint work together with Martin Feldhofer and Manfred Aigner. NFC stands for Near Field Communication and is one particular RFID standard [Int04a]. In the mCoupons application, a user can download coupons, for instance, from an advertisement or a poster, to a mobile device using NFC technology. These coupons are protected from copying or cloning, and no unauthorized person is able to alter the coupon. The security protocol for this application was based on AES and also on asymmetric cryptography for an extended version of the protocol. As the application of simulation software turned out to be favorable in previous developments, I designed an NFC simulator where the new protocol could be tested and refined before using it on a hardware device. The mCoupon system and the security mechanisms are described in detail in chapter 3 and also in [DA07] and [ADF07].

In order to extend the application range of the AES module, I did research on the feasibility and performance of symmetric authentication protocols for the EPC Gen2 communication standard [Int04c]. For this work I used an RFID simulator called *RFIDSim* to implement and evaluate the authentication protocols. The particular challenge of this development was not the authentication protocol itself, which at that time was already well known. The major part of my work was to get familiar with the simulator. *RFIDSim* is able to simulate even physical effects of the reader field. This feature is very useful for doing research on the physical layer, but was far too complex for our purpose, namely, to test the functionality and performance of an RFID protocol. Through the work with this simulator I gained invaluable insights about usability and requirements of RFID-protocol development tools, which were helpful to design the prototyping tool *ProtEx*.

At that time the IAIK *DemoTag* had already been used for several years at IAIK for prototyping and side-channel analysis. The IAIK *DemoTag* is an RFID tag emulator, which can be programmed to modify and extend its functionality. Brought into an RFID reader field, it behaves like a real RFID tag and supports

various RFID communication standards. First steps to control the communication between an RFID reader and the IAIK *DemoTag* over a software interface were taken by my colleague Michael Hutter. Based on the results of this work I developed the idea of an all-in-one prototyping tool. The basic concept of this tool is to join the existing simulators with a physical RFID reader and the IAIK *DemoTag*. Applications written for the simulator can be re-used to do application validation with physical devices. Additionally, software written to program the extended functionality of the tag can be re-used in the firmware of the IAIK *DemoTag* to provide the required features. I did research on an efficient design flow for secure RFID applications and designed the simulation and prototyping tool *ProtEx* to assist this development. Description of *ProtEx* in detail can be found in Chapter 4.

Another important topic I researched on is privacy for RFID systems. In order to show security mechanism involved to build privacy-preserving RFID applications, I defined a demo application using tagged pharmaceutical products. To protect the privacy of the owner, the tags can only be accessed by parties authorized by the owner. The protocol is based on reader authentication using asymmetric cryptographic primitives. The previously mentioned simulation and prototyping tool *ProtEx* was used to evolve the communication protocol, to implement software prototypes, and to develop and validate the hardware prototypes for the project. Details on the application have been published in [Dom11] and can also be found in Chapter 3.

In working on privacy mechanisms for RFID systems, authentication and confidentiality are applied to protect the transfer of sensitive data. In order to profit from security-enhanced RFID tags supporting these features, online access to these tags is the next evolution step. The lack of a clear concept on the connection of RFID tags to the *Internet of Things* motivated me to develop a new online-communication approach for passive RFID tags. This was joint work together with my colleagues Manfred Aigner and Stefan Kraxberger. The proposed system enables two-way communication with passive RFID tags using the Internet Protocol version 6 (IPv6). Full details of the communication concept can be found in Chapter 5. Moreover, the concept has been published in [DAK10] and [DGAK11]. In order to secure the end-to-end communication between correspondent nodes and RFID tags on the Internet, a security layer based on the proposed communication approach was defined. The security mechanisms involved to build this layer are based on Internet Protocol Security (IPsec) concepts. Description of the establishment of secure online communication for passive RFID tags can be found in Chapter 6 and in a publication in [DK11].

1.2 Design of the Thesis

The thesis consists of two parts. In the first part we focus on security protocols for RFID applications and the development of these applications. For this purpose, we analyze the current cryptographic capabilities of passive RFID

technology in Chapter 2. We give examples of how to integrate strong cryptographic algorithms in Chapter 3. There, we discuss the development and evaluation of sample RFID applications. The development process of secure RFID applications in general is discussed in Chapter 4. Furthermore, we show how the simulation and prototyping tools can speed up this process significantly.

The second part of the thesis deals with connection of passive RFID tags to the Internet. Chapter 5 presents a new approach to integration of passive RFID tags into the *Internet of Things* using standard Internet communication mechanisms (IPv6). The establishment of a security layer for the proposed communication protocol is described in Chapter 6. The following paragraphs outline the contents of the following chapters in more detail.

Chapter 2 analyzes the cryptographic capabilities of state-of-the-art RFID tags. We point out the current limitations for passive RFID technology, as passive RFID tags have rigid constraints in terms of power, time and area. Based on these figures, we present cryptographic hardware implementations that meet the constraints. The proposed hardware can be used to build security protocols for RFID systems. Furthermore, the chapter includes considerations on security in RFID systems in general. We elaborate security problems particular to RFID technology and discuss which cryptographic algorithms and protocols can be used as countermeasures to the identified security threats. We outline related work in this area and describe our contribution to the area of RFID security.

Chapter 3 deals with the development and evaluation of new security protocols for RFID technology. We describe different sample applications in order to show how the algorithms and protocols presented in Chapter 2 can be used to secure these applications. As security requirements vary depending on the application, different security measures have to be integrated into the presented applications. Furthermore, the use of simulation and prototyping tools for validation and evaluation of secure RFID applications is outlined.

Chapter 4 covers the design of secure RFID applications in general. We present a design flow, as a best-practice guideline, that involves various design steps from a first idea to the production of application-specific tags and readers. Particular attention in this design flow is paid to the development of security measures. Simulation and prototyping are considered crucial steps for a successful application design. In this chapter, we introduce simulation and prototyping tools that can be used to speed up these steps significantly. The described tools were developed during design of the RFID applications described in Chapter 3. We discuss structure and design goals, and show how the tools can be integrated into the design flow.

Chapter 5 is about integration of passive RFID technology into the *Internet of Things*. We describe an approach that enables passive RFID tags to communicate on the Internet. The proposed system is based on the mobility concept of the traditional Internet (Mobile IPv6). Thus, we describe basics of Mobile IPv6 and discuss modification to this protocol that are required for implementation on RFID tags. Prerequisites for RFID readers and tags to implement the proposed concept are identified. Most of the complexity of IPv6 communication is handled by the reader, whereas the tag only requires minor modifications. Furthermore, we describe how to handle particular scenarios, like offline communication or tag-triggered communication.

Chapter 6 focuses on secure end-to-end communication with passive RFID tags on the Internet. As security mechanisms are an indispensable part of a network connection, a security layer for the communication concept proposed in Chapter 5 is described. This security layer is based on the security protocol of the traditional Internet (IPSec). We present IPSec basics that are used to provide confidentiality, data integrity, and authentication. We discuss assumptions that have to be made for online communication with passive RFID tags. Furthermore, we show how the IPSec protocol can be adapted to the requirements of the proposed communication concept. Security considerations close this chapter.

Conclusions and an outlook on topics open to research can be found in Chapter 7.

Part I

Secure RFID Applications

2

Security for Passive RFID Technology

During the last years RFID has become an ubiquitous technology. It is used, for instance, in many different fields like supply chain management, ticketing, access control, health care, or animal tracking. The capabilities of RFID system have extended and application fields have expanded. For many new applications security has become an important feature. Cryptographic primitives are the fundamental building blocks for security protocols. In order to build secure RFID applications, research on feasible security algorithms for passive RFID tags is necessary.

Even before researching on cryptographic capabilities for RFID tags, we did research on efficient implementations of cryptographic hardware modules. Parts of this research have been published in [DM02], [Dom02], [MAD03], and [PMDW05]. The RFID technology is a new challenging platform for hardware security due to its very constrained resources. In this chapter, we give a survey on cryptographic primitives implemented in hardware, which are feasible for the power and timing constraints in passive RFID tags. In order to select the appropriate cryptographic primitive and the security protocol, the security requirements of an RFID application have to be identified. This starts with an analysis of potential threats to typical RFID applications and is followed by research on appropriate countermeasures to prevent these threats.

In this chapter, we give an overview on state-of-the-art implementations of cryptographic modules for RFID (see Section 2.1). Furthermore, we discuss security threats that have to be considered especially for RFID systems in Section 2.2. Countermeasures are presented in Section 2.3. We conclude the chapter with a discussion of standardized cryptographic primitives and protocols for providing strong security in RFID applications. Parts of the work presented in this chapter were published at the Cryptographic Hardware and Embedded

Systems conference 2004 [FDW04], at the 1st International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing 2005 [FAD05], at the Workshop on RFID and Leight-Weight Crypto 2005 [DOF05], and at the D.A.CH Mobility conference 2006 [DOF06].

2.1 Cryptographic Capabilities of RFID Tags

There is a large variety of RFID tags that differ significantly in their communication and computational capabilities. The Auto-ID Lab defines the following six classes of tags depending on their functionality. Table 2.1 depicts this classification. In our work we basically consider Class 2 tags, which are passive tags with advanced functionality. In the following we identify current constraints of these tags to be able to decide which cryptographic primitives are feasible for building secure RFID applications.

Table 2.1: RFID Tag Classes defined in [SE03]

Class	Description
0/1	read-only passive identity tags with basic capabilities.
2	passive tags with additional functionality like read/write memory or encryption.
3	semi-passive RFID tags. They may support broadband communication.
4	active tags. They may be capable of broad-band peer-to-peer communication with other active tag in the same frequency band, and with readers.
5	essentially readers. They can power other Class 1, 2 and 3 tags, as well as communicate with other Class 4 tags and with each other wirelessly.

2.1.1 Constraints of Class-2 RFID Tags

In general, we pay special attention to three constraints when designing hardware modules for passive RFID tags: Power consumption, timing, and size. The available power strongly depends on the operation-frequency range and the reading range of the system. Hardware primitives for long-range systems can count with a power budget of some hundreds of μWs, whereas hardware modules for close-coupling systems can consume about ten times more power. Power consumption is a very rigid constraint for RFID systems. Timing and size can be coped with more easily. These constraints depend mainly on limitations set, for instance, by the RFID communication standard or the manufacturer

of the tags. Timing problems can often be overcome by modifications of the communication protocol. The size of tags can in many cases be adapted to the application requirements, as new tags are often produced for the use in a particular application. In order to develop a working hardware module of any algorithm that should be used in RFID tags, each category of constraints has to be met. We describe how to deal with these limitations in the following paragraphs.

Power Consumption

Power is one of the main limiting factors for hardware modules on a passive RFID tag. The tag is powered by the reader field and only a part of the power dissipated by the reader can be gained by the tag. The longer the required reading range of the tag, the less power can be gained from the field, i.e., the more power a tag requires the closer it must be located to the reader. RFID systems can be categorized regarding their reading ranges; Close-coupling systems, like contactless smart cards, have a reading range up to 1 cm. Remote-coupling systems have a reading range between 1 cm and 1 m; all systems with longer reading ranges are called long-range systems.

The power dissipation of the reader is limited by legal regulations and can differ from country to country. The power distribution additionally depends on the frequency range the reader works in. Therefore, it is difficult to estimate an absolute value of the available power in an RFID tag at a certain distance to the reader. In [Fin03], an example of a typical energy dissipation in the near field ($<$ 1 m) of an RFID reader working at a frequency of 13.56 MHz (HF) can be found. According to [Fin03], an RFID tag located at a distance of 1 m from the reader has an available energy of 500 μW (@ 5 V). At a distance of 65 cm the available energy has already increased by a factor of 10 to 5 mW (@ 5 V). Taking losses into account, we expect all hardware modules with a power consumption less than 1 mW to be feasible for remote-coupling systems. For close-coupling systems we calculate with a power budget of about 5 mW.

For UHF readers, the effective radiated power (ERP) is limited to 2 W in Europe, and to 4 W in the USA. The power decay in the far field is indirectly proportional to the square of the distance and is also indirectly proportional the square of the frequency. In [CDDJ] the authors present a UHF tag design which has a reading range of 12 m. The formula the authors use to calculate the power distribution in the far field is as follows:

$$P_{AV} = P_{EIRP} \cdot G_R \cdot \frac{\lambda^2}{(4\pi d)^2}.$$

P_{AV} denotes the power available at the tag. The equivalent isotropically radiated power (EIRP) is denoted with P_{EIRP} and has the value of 4 W in the considered system. The antenna gain (G_R) is set to 1 (= 0 dB), which means that the tag antenna is considered as lossless for this calculation. The operation frequency is 2.45 GHz, the wavelength λ is therefore 0.1224 m. d denotes the distance of the tag from the reader. Using the formula, a tag with an antenna

gain of 0 dB can harvest about 380 μW @ 1 m, about 15 μW @ 5 m, and about 3.8 μW @ 10 m. We base our estimation for the available power in UHF tags on these values. When taking losses in the radio field and at the tag antenna into account, the available power is smaller than the calculated values. As a rule of thumb we consider hardware primitives that require less than 100 μW of power (which corresponds to a reading range of about 2 m, using the formula) as feasible for long-range RFID systems.

Timing

RFID communication standards define a maximum response time for the tag. This means that, the time frame in which the tag has to respond to a reader request must not exceed a certain limit. The response time is specified in the physical layer of the communication standard used. For the HF protocol specified in the ISO-15693 standard [Int01], which corresponds to the ISO-18000-3 standard [Int04b], and the UHF protocol ISO-18000-6 alias EPC-Gen2 ([Int04c]) the response time is about some hundreds of μ-seconds (e.g., $\sim$ 320 μs for ISO-18000-3). For the HF standards ISO-14443 and ISO-18092 ([Int00], [Int04a]) the response time can be up to 4.95 seconds. In these standards, a dedicated *waiting-time-extension* process has to be performed to extend the response time to this value.

In many cases the tag response time is too short to perform complex cryptographic operations within this time. To overcome this problem we use a concept called *interleaved protocol execution*. This concept works for most of the existing RFID communication standards and was first published in [FDW04]. Figure 2.1 illustrates the principle of interleaved protocol execution. We propose that a reader sends a request to the tag without expecting an immediate response to this request. Nevertheless, the tag can respond with an affirmation that the request was received correctly. This immediate response does not contain the requested calculation result. Afterwards the reader starts polling for the response from the tag. The time between sending the request and polling for the response can be used to send requests to other tags. If more than one tag is present in the field, the communication protocol can be processed in parallel for different tags. In this way, the performance loss, caused by the separation of sending the request and polling for the response, can be limited.

In the interleaved protocol mode, the reader sends a challenge to the tag and polls for the response afterwards. The protocol shown in Figure 2.1 involves three tags. `C1` denotes the challenge for tag 1, `C2` denotes the challenge for tag 2 and so on. After tag 1 has received the challenge, it starts calculating the response (`R1`). In the meantime, the reader sends challenges to tag 2 and to tag 3. Afterwards, it requests a response from tag 1 (`Rec R1`). If the tag has already finished the calculation, it sends the response (`Resp R1`). Following this concept, the calculation of cryptographic operations, which are often very time consuming, are feasible for RFID communication standards where the calculation time exceeds the tag response time.

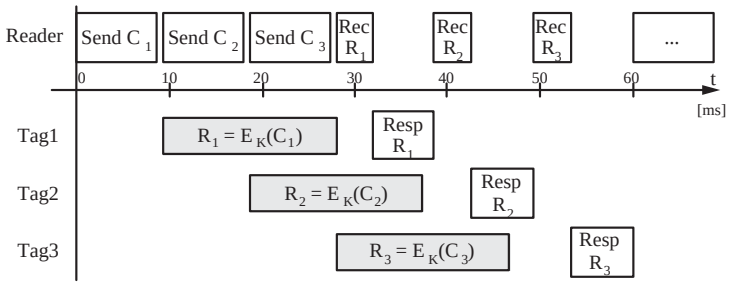


Figure 2.1: Interleaved Protocol, Picture taken from [FDW04]

Size and Memory

Apart from power and timing being the most decisive constraints, size in hardware has as well to be considered when designing cryptographic hardware modules for passive RFID tags. The size of the tag hardware can be limited, for instance, by spacial requirements of the application, limitations posed by the manufacturer of the tags, or by financial matters. The larger the size of the tag's hardware, the more expensive the tag. The application designer has to communicate the requirements for the tag used in the application. The manufacturer will provide the appropriate platform for these requirements. As for new applications also new tags will be produced, the size of the tags can be adapted to the needs of the application. The hardware of secure tags will be more extensive because of the additional cryptographic primitives needed. This will impose higher costs on the application. Nevertheless, if security is a strong requirement, these costs are inevitable.

Current RFID tags offer up to 64 KBytes of memory [Fin03]. In general, this memory size meets the memory requirement for the calculation of even complex cryptographic algorithms, like ECDSA. This means that one single calculation of such an algorithm can be performed on the tag without running out of memory. Nevertheless, an entire security protocol consists of a series of calculations of various cryptographic algorithms and storing intermediate values. The execution of a security protocol needs additional memory for storing intermediate data, large keys, or certificates. Therefore, when performing a complex security protocol, memory limitations could be a problem. Although we expect memory availability of RFID tags to rise over time, each detail of the protocol has to be analyzed for its memory requirements to decide whether it is feasible for the proposed RFID tags.

2.1.2 Cryptographic Primitives

Cryptographic primitives are the building blocks of every security protocol. The hardware implementation of some of these primitives is complex and their calculation is time consuming. As passive RFID tags have limited resources, it is a challenge to find appropriate hardware primitives for this platform. The

RFID community follows two approaches: the first one searches for new cryptographic primitives, which are "light-weight" enough for RFID tags. The second approach tries to find new "low-power" implementations for well-established cryptographic algorithms. We propose to build security protocols based on standardized cryptographic algorithms and protocols. This approach offers a high level of security, as the security features of standardized algorithms are in general well-tested by many different communities. Furthermore, we intend to support interoperability, which is more likely to achieve with standardized protocols.

In order to provide the basic security services confidentiality, authentication, data integrity, and non-repudiation, the following categories of algorithms are required: Encryption algorithms, hash algorithms, and digital signature algorithms. For each category, hardware implementations for passive RFID technology already exist, which meet the constraints identified in the previous section. In the following we present the most promising publications for each type of primitives.

Symmetric Encryption

Symmetric encryption algorithms provide confidentiality and can be used for authentication. Each participant owns the same secret key. The knowledge of the key proves the authenticity of the owner. Authorized parties can encrypt messages by using the secret key. Confidentiality is provided as only the owner of the secret key can decrypt encrypted messages and reveal its content. Authentication can be provided by using a challenge-response protocol: The verifier sends a challenge, which is encrypted by the prover. Only if the prover knows the secret key, it can produce the correct response to the challenge. If the verifier receives the correct response, it authenticates the prover.

The first hardware implementation of the *Advanced Encryption Standard* (AES) [Nat01] for the use in passive RFID tags has been published in [FWR05]. This module has a size of 3,400 gate equivalents (GE). It needs 4.5 μW of power and about 10 milliseconds to perform an AES calculation at a frequency of 100 kHz. Another promising work in this area is the implementation of Kim [KRCJ06]. This hardware module is a bit larger than the first one, it requires 3,900 GE. It needs 4.9 μW of power and about 9 milliseconds to perform an AES calculation. Considering the constraints described in Section 2.1.1, both

Table 2.2: AES Primitives I - Power consumption

Reference	Technology [μm]	Supply [V]	Power [μW]	Current [μA]
[FWR05]	0.35	1.5	4.5	3.00
[KRCJ06]	0.25	2.5	4.9	1.94

Table 2.3: AES Primitives II - Size and Timing

Reference	Size [GE]	Freqency [kHz]	Cycles	Time [msec]
[FWR05]	3,400	100	1032	10.3
[KRCJ06]	3,900	100	870	8.7

implementations are feasible for passive RFID tags. The two hardware modules are compared in detail in Table 2.2 and Table 2.3.

Hash Algorithms

Hash algorithms are used to produce "fingerprints" of a message. They map a message of unlimited size to a hash value with fixed size. This mapping works only one-way: It should be infeasible to generate the original message only from knowing the hash value. Furthermore, it should be hard to find two messages that map to the same hash value. If these conditions are met, the hash value can be treated as a representative of the message. Hash algorithms can be part of digital signature algorithms, which are described in the next paragraph, and can be employed to build message authentication codes (MAC). MACs provide data integrity, this means that they are used to check whether a message has been modified or not.

The most common hash algorithms used nowadays are the *Secure Hash Algorithm* (SHA) families 1 (SHA-1) and 2 (SHA-224, SHA-256, SHA-384, SHA-512). Both families are specified in [Nat02]. Table 2.4 and Table 2.5 show the results of five different works on low-power implementations of SHA-1 and SHA-256. The size of the hardware modules ranges from 5,527 GE to 10,868 GE. The calculation time can be about 3 milliseconds for [O'N08] up to about 12 milliseconds for [FW07]. The most rigid constraint for passive RFID tags is the

Table 2.4: HASH Primitives I - Power Consumption

Algorithm	Reference	Technology [μm]	Supply [V]	Energy [μW]	Current [μA]
SHA-1	[FR06]	0.35	3.3	35.20	10.68
SHA-1	[FW07]	0.35	1.5	5.90	3.93
SHA-1	[O'N08]	0.18	1.8	13.80	7.70
SHA-1	[O'N08]	0.13	1.2	2.32	1.90
SHA-256	[FR06]	0.35	3.3	52.40	15.87
SHA-256	[FW07]	0.35	1.5	8.80	5.86

Table 2.5: HASH Primitives II - Size and Timing

Algorithm	Reference	Size [GE]	Frequency [kHz]	Cycles	Time [msec]
SHA-1	[FR06]	8,120	100	1,247	12.5
SHA-1	[FW07]	8,120	100	1,247	12.5
SHA-1	[O'N08]	6,122	100	344	3.4
SHA-1	[O'N08]	5,527	100	344	3.4
SHA-256	[FR06]	10,868	100	1,128	11.3
SHA-256	[FW07]	10,868	100	1,128	11.3

power consumption. The presented hash algorithms require from 5.9 μW up to 52.4 μW of power. These power consumptions meet the constraints even for long-range systems.

In 2007, the *National Institute of Standards and Technology* (NIST) issued a call for competition to find a new standardized hash algorithm, as the SHA-1 algorithm had already shown some weaknesses (like described in [WYY05]). The main objective of the competition was to find a new hash primitive, which would be defined as the SHA-3 algorithm and would extend the existing SHA standard. The candidates were encouraged to show that the possible hash algorithm could be effectively implemented in hardware. This is also an important point for implementing such algorithms on RFID tags. At the moment of writing this thesis, the five final candidates are under review. The decision will be taken in 2012.

Digital Signature Algorithms & Asymmetric Encryption

A digital signature algorithm is used to provide authentication, non-repudiation, or confidentiality. Digital signature algorithms work with asymmetric keys, this means that each system participant owns a key pair consisting of a public and a private key. The private key must be kept secret, the public key can be distributed. To provide authentication, a challenge-response protocol is performed: The owner of the private key signs a challenge with its private key. The prover verifies the signature by using the public key of the signer. If the signature is valid, the prover authenticates the verifier. Furthermore, the signer cannot deny having signed the message, which corresponds to the security service of non-repudiation. If a sender wants to encrypt a message for only one particular receiver, the sender can encrypt the message using the public key of the receiver. In that way, only the receiver is able to decrypt the message with its private key. In this way, confidentiality is provided.

Asymmetric cryptographic algorithms are very demanding primitives in terms of size, power and time. The use of *Elliptic Curve Cryptography* (ECC) allows

Table 2.6: ECC Primitives I - Power Consumption

Algorithm	Reference	Techn. [μm]	Supp. [V]	Energy [μW]	Current [μA]
GF(P192)	[Wol05]	0.35	3.3	30.00	9.10
GF(P192)	[Aue08]	0.35	2.5	613.70	245.46
GF(2^{191})	[Wol05]	0.35	3.3	30.00	9.10
GF(2^{163})	[HWF08]	0.35	2.5	54.70	21.88
GF(2^{163})	[HWF08]	0.18	1.8	10.80	6.00
ECDSA GF(P192)	[Aue08]	0.35	2.5	497.47	199.00
ECDSA GF(P192)	[HFP10]	0.35	3.3	1,277.00	387.00
ECDSA GF(P160)	[KF10]	0.35	3.3	860.00	260.60
DH GF(2^{163})	[BBD$^+$08]	FPGA	3.3	79.00	23.90

to reduce this effort significantly. Therefore, most of the public-key algorithms proposed for the use in RFID tags base on ECC. One very common digital-signature algorithm based on ECC is the *Elliptic Curve Digital Signature Algorithm* (ECDSA), which is specified in [Nat00]. ECDSA can be performed with so-called point operations on elliptic curves. Some of the implementations we consider for use in RFID tags only provide these operations. Here, the ECDSA has to be composed from the single operations. Other hardware modules provide the calculation of the complete algorithm in one step. Table 2.6 and Table 2.7 give an overview of low-power implementations of ECC primitives.

The proposed solutions offer different features. Some hardware modules only provide point operations, like [Wol05], [Aue08], or [HWF08]. The point operations are furthermore performed on different arithmetic fields (GF(P192), GF(2^{191}), and GF(2^{163})). Other implementations provide the calculation of a complete cryptographic algorithm like ECDSA ([Aue08], [HFP10], [KF10]) or Diffie-Hellmann Authentication ([BBD$^+$08]). [HFP10] and [BBD$^+$08] also include a random number generator (RNG), which is also an essential service for building security protocols.

The benchmarks of the hardware modules are as different as the features they provide. The size of the implementations ranges from about 13,000 GE to nearly 25,000 GE. The amount of time required for one calculation differ significantly. The fastest module needs 95 milliseconds (for performing a DH Authentication), whereas the slowest implementation takes 11 seconds for an elliptic-curve scalar multiplication. As we have shown in Section 2.1.1, timing issues can be met by using interleaved protocols. Therefore, even a calculation time of various seconds is acceptable for the use in an RFID communication protocol. Power consumption of the different modules range from 10.8 μW to

Table 2.7: ECC Primitives II - Size and Timing

Reference	Features	Size [GE]	Frequency [kHz]	Cycles	Time [msec]
[Wol05]		23,800	60	677,500	11,450
[Aue08]		24,745	1,000	953,854	953
[Wol05]		23,800	60	426,300	7,100
[HWF08]		13,320	106	296,000	2,960
[HWF08]		13,250	106	296,000	2,960
[Aue08]		24,745	1,000	1,030,656	1,030
[HFP10]	SHA-1, RNG	19,115	847	859,188	1,014
[KF10]		18,247	1,000	511,864	512
[BBD+08]	DH, RNG	12,876	847	80,465	95

1,277 μW. This means that the most demanding hardware modules in terms of power are not applicable for long-range RFID systems, where the power consumption of an RFID tag should be less than 100 μW to provide an acceptable reading range. Nevertheless, these modules can be used in short-range and close-coupling systems. We have described the power limits for the different RFID systems in Section 2.1.1.

Due to the different features they provide, the different implementations do not compare. An application developer has to consider the application requirements to select an appropriate cryptographic module. The security requirements of an application strongly depend on the security threats that are relevant for the application. Therefore, a threat analysis has to be done before the decision on the cryptographic algorithm can be taken. In the next section we discuss potential security threats when using RFID technology.

2.2 Security Threats

As RFID applications are manifold and are thus susceptible to several security risks. RFID technology works "non-interactive". This means that RFID tags may communicate with readers without the user even noticing it, which implies a security risk per se. RFID works also "non-line-of-sight" which enables the attacker to act remotely without any physical contact to the tag. These properties cause typical security threats that are to be considered in development of RFID systems. for RFID systems can be identified. In this section, we identify three main threat categories, namely tracking and tracing, cloning and counterfeiting, and unauthorized access to the tag. The first approach to this categorization has been published in [DOF06]. Inputs from other authors (e.g.,

[GJP05], [Jue06], [Lan09a], or [SE09]) have been considered during development of a more detailed categorization, which is presented in this section.

Speaking of RFID security, one of the main concerns is privacy. This topic has been discussed since the beginning of RFID technology and is still present in the media. Privacy can be defined as a person's right to control access to his or her personal information. This means that, the owner of information can define authorized parties with access rights. All others have to be excluded from access. We define three requirements that should be fulfilled for an optimal control of a person over her information:

- The person should be aware of the access to her data.
- The person should be able to accept or decline the access.
- The person should be able to stop or launch an access to her data.

Two types of privacy can be distinguished: Data privacy and location privacy. In the first case, sensitive data has to be protected against unauthorized access. In the second case, the unauthorized person should not be able to determine the current or past location of a person or item. These two types of privacy are treated separately in the presented categorization.

In this section, attacks and threats on the application layer are considered. Attacks on the physical layer like implementation attacks or side-channel analysis are out of scope. When dealing with RIFD, three major security threats are identified: Tracking and Tracing, Forgery and Counterfeiting, and Unauthorized Read/Write Access to the tag's memory. In the following we explain these threats in detail.

2.2.1 Tracking and Tracing

As unprotected tags reveal their identity to every standard reader, an attacker can easily find out the identifiers (IDs) of items. Information about valuables and goods a person has on her can easily be used to victimize a person or to draw a precise picture of the person's life. Things like the current location or personal preferences can be acquired from these data. Langheinrich sees this aspect also as one of the novel privacy challenges ubiquitous computing is confronted with [Lan09b], [FM05]. In addition to the temporal and spacial expansion of the data acquisition, the author describes a change of quality in data acquisition: Readers vanish in the environment and the user is no longer aware that information is exchanged.

If items can be associated with a person, this person can be identified when moving from one reader field to another. This scenario is called tracking and affects the location privacy of a user. Another problem arises if tracking data is stored in a database and the past locations of an item (and the assigned user) can be accessed by unauthorized parties. This scenario is called tracing. The public discussion about tracking and tracing has created a bad image of RFID technology in the past. To get a broad public acceptance of this technology, it

will be essential to address this topic. Also, the European Commission states that *"RFID will only be able to deliver its numerous economic and social benefits if effective measures are in place to safeguard personal data protection, privacy and the associated ethical principles that are central to the debate on public acceptance of RFID"* [Com09].

2.2.2 Cloning and Counterfeiting

RFID tags are used to identify items. This property is used in access systems (e.g., car immobilizers, ticketing) as well as for proof-of-origin applications. The mere existence of tags attached to a product complicates counterfeiting. If an attacker is able to counterfeit or even clone the RFID tag attached to the product, the proof-of-origin property of the tag vanishes. Especially for access systems, forged tags can cause a serious damage. For performing such an attack, the attacker monitors the communication of existing tags or even has physical access to a tag. From the data gained through this monitoring the attacker can build her fake tag. The difference between cloning and counterfeiting is, that cloning produces a fake tag that pretends to be an already existing tag, whereas counterfeiting produces a fake tag that generates new data to pretend to belong to an authorized group of tags.

A special type of counterfeiting is a relay attack. In this attack, the attacker has one or more connected devices that have access to both the reader and an original tag. One device plays the role of the reader for the tag (fake-reader device), and one device plays the role of the original tag for the reader (fake-tag device). Fake-reader and fake-tag device are connected, either wireless or by cable. The fake-reader device relays the messages from the original reader to the fake-tag device. The fake-tag device sends the received messages to the original tag, which responds to the message. The response from the original tag is relayed via fake-tag and fake-reader to the original reader. In this way, an attacker can impersonate the original tag.

2.2.3 Unauthorized Read/Write Operations

At the beginning of this section we have defined privacy as the user's right to control the access to her data. The European Commission states that: "*Application of cryptographic primitives on tags as privacy enhancing technology (PET) to protect personal data on tags is suggested.*" [Eur05]. When considering sensitive data (like health or financial data) the threat scenario is obvious: If an attacker gets access to sensitive data, she can use it to the disadvantage of the tag's owner. An employer might, for instance, take advantage of the information about one of her employee's serious disease or pregnancy. Another scenario could be a thief who gets information about the value of a product a person has on her. Therefore, read operations should only be granted to authorized readers in certain applications. Unauthorized write access to a tag can also have negative consequences: If an attacker could modify the data stored on the memory, the tag could be invalidated (e.g., in ticketing or access systems).

If the attacker is the owner of the tag, she can change the data to her favor, for instance, by adding extra value in ticketing systems.

In many applications, only authorized readers should have access to the tag's data. The owner of the tagged item and/or the manager of the tag should have the possibility to manage these access rights. Each time a read or write command is being performed the tag is supposed to check the authorization of the reader. Another service which has to be provided when handling access rights is the possibility to transfer the control over the tag to a new owner. This service is called transfer of ownership.

2.3 Meeting Security Requirements

In the previous section we have defined different types of threats for RFID systems. In this section we discuss a set of countermeasures that can be used to handle these threats. Various protocols have been proposed to address RFID security in the past few years. The building blocks of security protocols are cryptographic primitives. As we have mentioned in Section 2.1.2, there are two different approaches when searching for cryptographic primitives for RFID: Using standardized algorithms or designing proprietary light-weight algorithms. In the following we map the proposed algorithms and protocols to the security threat they address. In our security protocols, we exclusively use standardized algorithms in order to provide a high level of security and interoperability. Therefore, we finish this section by giving an outline, how standardized cryptographic protocols can be used as countermeasures for the different types of threats.

2.3.1 Tracking and Tracing

The property which has to be provided by a security protocol to prevent tracking and tracing is location privacy. This means that no unauthorized party is able to locate the tag (the user, respectively), neither in the present nor in the past. This requirement can be met, if the tag does not send its identifier to unauthorized readers. Another measure to "hide" the identity of a tag is to change its identifier over time.

Many approaches have been published to prevent tracking and tracing. The first group of countermeasures disable the tag physically. In this case, the tag is not able to talk to any reader, not even to authorized ones. A kill command authorized by a password can permanently deactivate the functionality of the tag. The same effect is reached by ripping off the tag from the item. These measures prevent the access to the tag permanently but not only for unauthorized parties but also for the owner, who cannot take advantage from the RFID functionality after deactivation. Another physical measure of location-privacy protection is shielding of the tag while it should not be read. This measure presumes awareness of the owner whether the tag has to be protected or not in different situations. A similar effect can be reached by using the so-called

blocker tag proposed by Juels [JRS03]. The blocker tag is able to simulate a whole range of IDs and jams the inventory procedure of an unauthorized reader. In that way, the reader cannot determine the ID of the "real" tag.

Other proprietary approaches that address the tracking threat, are, for example, the hash-lock schemes published in [WSRE03] and [JW06]. In these schemes, the tag sends its ID only to readers knowing a particular key, which is unique for the tag. The key is searched in a database on basis of the hash value the tag sends as response to the inventory request. A similar approach, which also uses hash functions and a large back-end database, has been suggested by NTT Labs [OSK03]. Other pseudonym schemes based on a key-search in trees have been published in [MW04] and [MSW05].

Further approaches to prevent tracking and tracing are silent tree walking (used for a particular anti-collision algorithm) [WSRE03], one-time pad schemes (where the tag and the reader have to exchange lists of one-time pads) [Jue04], global and private IDs, or light-weight authentication protocols [VB03]. Floerkemeier et. al., Rieback et al., and Juels et al. suggest methods, where a mobile device different from the RFID tag intermediates the reader requests and collects information from the reader. This mobile device acts as a guardian for the tag and controls the information flow for the protected tags [FSL04], [RCT05], [JSB05].

The algorithms mentioned above represent a selection of light-weight implementations providing location privacy. The list of proprietary countermeasures to tracking and tracing is very long. In difference to the light-weight security approach, we address the tracking threat using standardized cryptographic algorithms. We suggest the use of random identifiers (= pseudonyms) for this purpose. The tag requires a random-number generator to provide this functionality. Each time the tag enters a reader field, it generates a new pseudonym and uses this identifier in the inventory procedure. Afterwards, the reader has to authenticate before the tag reveals its ID. We first presented this approach in [DOF05].

Authentication in our approach is performed using standardized algorithms and protocols. Reader authentication can be obtained by different means, which we describe in Section 2.3.4. The proposed protocol is a special case of an authenticated read operation on the tag. We handle this scenario in detail in Section 2.3.3. Another point to be observed is the security of the communication channel even to authorized readers. An eavesdropper should not be able to perceive the transmitted ID. Otherwise, the use of pseudonyms in the inventory procedure is useless. Therefore, we suggest to encrypt further communication between the authorized reader and the tag.

2.3.2 Cloning and Counterfeiting

The property a security protocol should offer to prevent cloning and counterfeiting is the proof of origin of the tag. Only tags that originate from an authorized party can perform this proof. This requirement can be reached by tag authentication. Tag authentication works like reader authentication, but the roles of the

parties have changed: The tag is the prover, and the reader is the verifier. We describe authentication protocols for RFID systems in detail in Section 2.3.4.

A hardware approach to prevent cloning and counterfeiting are *Physical Unclonable Functions* (PUFs). The idea that physical properties of a hardware device can be used to identify this device uniquely has been published in [Pap01]. Proposals how this technique can be used for RFID tags can be found in [REC04], [BR07], [DSP+08] or [SVW10]. As this approach is only applicable on the physical layer, it is out of scope of our considerations of countermeasures on the application layer.

In Section 2.2 we have defined the relay attack as a special case of counterfeiting. The most effective countermeasure against relay attacks seem to be *Distance Bounding Protocols*. These protocols measure the round-trip time of the radio signal and define an upper bound for the distance between the reader and the tag. As the signal takes some additional time to propagate through the transmission line between fake-reader device and fake-tag device, a relay attack can be detected. Various distance bounding protocols have been proposed over time, for instance, [HK05], [MOP06], [RNTS06], [AT09], [Han10], or [KKBD11]. We do not further consider these protocols, as the proposed protocols do not exclusively work on the application layer.

A very interesting approach for tag authentication is the *Direct Anonymous Attestation* (DAA). This algorithm combines a proof of origin with providing location privacy. Basically, a party can prove to be a member of an authorized group, but does not reveal the particular identity. It was first published in [BCC04] and has been adopted by the Trust Computing Group for remote anonymous attestation of *Trusted Platform Modules* (TPMs). DAA is computationally very expensive and is at the moment not suitable for today's low-end devices on the *Internet of Things*. Efforts to reduce the resources for DAA and studies on the feasibility of DAA for RFID systems are ongoing.

2.3.3 Unauthorized Read/Write Operations

The property a security protocol has to provide to prevent unauthorized read and write operations on the tag is access control. The reader has to prove that it is allowed to access some of the tag's data. The tag verifies access rights. To obtain access, the reader has to authenticate to the tag. Reader authentication can be done using different mechanisms. In the last years, many so-called "lightweight" mechanisms have been proposed to omit the computational overhead of standardized cryptographic primitives. In the following, we give some examples of proprietary reader-authentication schemes. Some of them have already been mentioned in section 2.3.1, as location privacy can also be reached by reader authentication.

The so-called hash-lock scheme [WSRE03] proposes that tags send the hash value of their secret key to the reader. The reader can find out the corresponding key by searching a data base and prove the knowledge of this key. In that way, the reader is authenticated. If always the same hash value is transmitted, tracking of the tag is possible. Therefore, this scheme is also available in a

randomized variant. Ohkubo proposes an improvement of this scheme by using two hash functions and building a hash chain [OSK03]. Molnar and Wagner propose a tree-based key search in [MW04], i.e., each tag stores a set of keys organized in a tree. Only authorized readers know the key tree and can therefore derive the correct key in a challenge-response scheme.

Another approach to access control for RFID tags are agent schemes. In these schemes an agent device mediates the communication between reader and tag. Juels suggests the use of so-called *proxy devices* which can presume the identity of all supported tags and acts on behalf of these tags in respect to the reader [JSB05]. One advantage of using agents is the computational power of agent devices. As the agent has in general more computational power than a tag it can easily perform even elaborated cryptographic protocols and take over complex calculations instead of the tag.

Rieback et al. suggest another agent scheme, which does not require additional functionality on the tag [RCT05]. They present a privacy guardian with a security policy which lets the guardian selectively allow or jam different reader-tag communications. The security policy determines which reader, in which situation, has access to which tag. If a reader is not authorized, the communication with the tag is jammed. When using agent schemes, the owner of the tags has to decide wether a communication is established or refused. Configuration of this policy should be easy for the user.

In all of the mentioned methods the reader has to prove that it is allowed to access the tag's data. This process is called reader authentication. In difference to the approach of using light-weight or proprietary algorithms for reader authentication, we follow the approach of using standardized cryptographic primitives and protocols for this purpose. As we base the development of secure RFID applications on this approach, we describe methods of standardized authentication in detail in the next section.

Additional security services that has to be provided as regards access control is the transfer of ownership as well as confidential message exchange. Transfer of ownership can in general be provided by key-management functions, as in most applications authentication is performed by showing the knowledge of a key. The key of the new owner has to be transferred to the tag. This operation should only be performed by an authorized party (former owner, tag manager).

Confidential message exchange can be provided by encryption. Confidentiality is an issue for authorized read operations on the tag, because the requested data have to be somehow transmitted to the authorized reader. An unauthorized reader that is located near to the authorized reader is able to sniff the communication between the RFID components. Although the unauthorized reader cannot control the type of information it gets, this scenario should also be considered as, for instance, the location privacy can be corrupted if the identity of the tag leaks. Therefore, we propose to encrypt the communication to hide the transmitted data from unauthorized access.

As authentication is an essential countermeasure for each of the discussed security threats, we describe how authentication is provided by standardized

algorithms and protocols in the next section. Authentication can either be provided by symmetric or asymmetric cryptographic primitives. We compare symmetric and asymmetric approaches and describe standard key-management methods.

2.3.4 Standardized Protocols for RFID Security

In the last section we have presented an overview of various countermeasures to handle security threats in RFID technology. These countermeasures also contain proprietary and light-weight algorithms. In this section we focus on standardized cryptographic protocols. Table 2.8 lists security threats, assigned security requirements, and the proposed security services. The security threats to RFID systems addressed in the table are described in Section 2.2. A security requirement is the property of a security protocol that has to be provided to prevent a particular threat. A security service is a mechanism that can be used to meet the security requirement. Each security service (e.g., reader authentication) can be implemented by different algorithms and protocols.

To provide location privacy we suggest to use pseudonyms in the inventory process. Each time the tag enters a new reader field a random number is generated which is used as identifier for this reader session. As every time a new pseudonym is used, this number does not uniquely identify the tag, and the tag can no longer be tracked. To provide this service a *(Pseudo) Random Number Generator* (PRNG) is required. Implementations of PRNGs for RFID can be found in some hardware modules presented in Section 2.1 where the PRNG functionality comes as part of another algorithm ([HFP10] or [BBD+08]). Other suggestions for implementing a PRNG on an RFID tag can be found, for instance, in [GJR07] or [PLHCETR07]. For most applications, readers has to reveal the identity of the tag, which is prevented when using pseudonyms in the inventory process. Therefore, an authorized reader uses a dedicated command to request the identifier of a tag. By using encryption for the transfer of the ID, sniffing of another (unauthorized) reader can be prevented.

Cloning and counterfeiting of a tag can be prevented by a proof of origin of the tag. The tag proves to the reader that it knows a secret (key) which is only known by authentic tags. This means that the tag has to authenticate to

Table 2.8: Security Threats, Requirements and Services

Threat	**Security Requirement**	**Security Service**
Tracking and Tracing	Location Privacy	Pseudonyms Reader Authentication Encryption
Cloning and Counterfeiting	Proof of Origin	Tag Authentication
Unauthorized Read/Write Access	Access Control	Reader Authentication Encryption
	Transfer of Ownership	Key Management

the reader. Unauthorized read/write accesses to the tag can also be prevented by using authentication. In this scenario, the reader has to authenticate to the tag. The tag reveals its information only to authorized readers. For communication between authorized parties, encryption should be used to prevent sniffing. Transfer of ownership is also a requirement for authenticated operations. If the owner of the tag changes, the former owner should transfer the control over all access rights to the new owner. This requirement can be met by setup of key-management mechanisms on the tag. The previous owner can, for instance, authenticate the key of the new owner on the tag.

It is evident that authentication is one central security requirement for RFID security. In the following we describe how authentication can be provided by using standardized cryptographic algorithms and protocols. In an authentication protocol two parties are involved: the prover (P) and the verifier (V). The prover proves its identity, the verifier checks this proof. The prover owns a secret (key) and proves this ownership to the verifier. The proof is in general performed using a challenge-response protocol, which works as follows:

$$\begin{aligned} V &\rightarrow P \quad : \quad C \\ P &\rightarrow V \quad : \quad Enc(C)_K \end{aligned}$$

The prover claims its identity. The verifier sends a challenge C to the prover. This challenge is random and not predictable. The prover performs a cryptographic operation (Enc) on this challenge using its secret key (K) to generate a response and sends the response to the verifier. The verifier checks the prover's response and if it is correct, the prover's identity is treated as authenticated.

For performing challenge-response protocols, symmetric as well as asymmetric cryptographic primitives can be used. Symmetric challenge-response authentication is standardized in [Int99]. In this scenario, both parties hold the same secret key. The prover encrypts the challenge with the key, the verifier can prove the response by encrypting the challenge or by decrypting the response using the same key. Asymmetric challenge-response authentication is standardized in [Int93]. In this scenario the prover owns a key pair consisting of a public and a private key, whereby the public key is shared with the verifier. The prover signs the challenge with its private key, the verifier can prove the response by verifying the signature with the public key of the prover. As symmetric primitive for RFID systems we suggest to use the *Advanced Encryption Standard*(AES), specified in [Nat01]. As asymmetric primitive we suggest the *Elliptic Curve Digital Signature Algorithm* (ECDSA) specified in [Nat00].

We have already presented cryptographic primitives implemented in hardware for the use in passive RFID tags in Section 2.1. It is obvious from the performance figures, that symmetric algorithms are more power-efficient and less time-consuming than asymmetric algorithms. The drawback of symmetric algorithms lies in the key management: Each participant keeps the same key, which has to be kept secret. Depending on the application, there will be a lot of devices holding the same key. If one of the devices leaks the key, the whole system is corrupted. Generation and particularly the distribution of a new key

to all participants is complicated. For this reason, symmetric primitives are mainly used in closed-loop applications, where one central instance has control over all participants.

Key management is easier when using asymmetric primitives. Each participant owns a key pair consisting of public and private key. One particular private key is stored exclusively on one device and is kept secret. The public keys of all participants are published. With a public key, a signature generated with the corresponding private key can be verified. It is also possible to "encrypt" a message with the public key, which can only be "decrypted" with the corresponding private key. In this way private messages can be addressed to only one particular device in the system. If one device is corrupted the other participants are not concerned, because each device only holds its own private key. Furthermore, scalability for a system using asymmetric primitives is easy to provide: A new participant generates its public/private key pair and publishes the public key. Due to the easier key management and scalability, asymmetric or public-key cryptography is often used for open-loop applications.

A common practice when using asymmetric primitives is the establishment of a *Public Key Infrastructure* (PKI). A PKI is a hierarchical trust model with a *Certification Authority* (CA) on top. The CA is trusted from all participants and can sign public keys from participants in the system. This signed public keys are called certificates, which also hold additional information about the participant and the CA. If one participant wants to verify the authenticity of the public key and/or identity of another participant, she can verify the certificate by proving the signature of the CA. The public key of the CA is in general well-known or easy to retrieve.

The knowledge of the public key of the CA is the only requirement to do a verification of a certificate and a signature, therefore such systems are often used for offline applications. An offline system works as follows: Each participant has stored the public key of the CA and can therefore verify all certificates issued by the CA. A certificate contains the public key of another participant in the system. If the certificate is valid, the public key can be treated as authenticated and can be used to perform a secure protocol without accessing an online database. The hierarchy of a PKI can be very flat, if there is only one

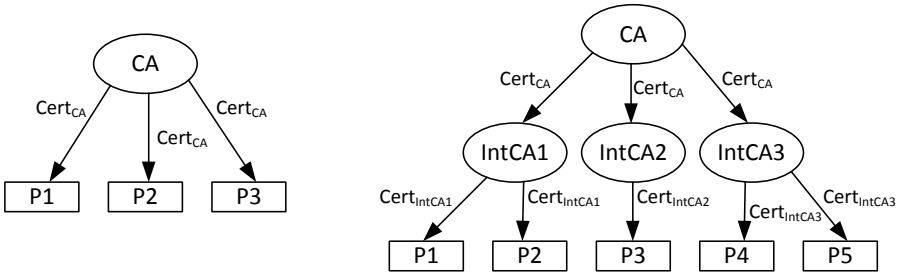


Figure 2.2: Public Key Infrastructures

CA which issues all certificates of the participants. The hierarchy can also be very steep, if intermediate CAs are used. Intermediate CAs require a certificate from a CA which belongs to a higher hierarchy level and can issue certificates for other intermediate CAs or participants belonging to a lower hierarchy level. Figure 2.2 shows the principle of PKIs with and without intermediate CAs.

Symmetric as well as asymmetric authentication protocols use challenge-response mechanisms for proving the knowledge of a secret or private key. Symmetric algorithms offer a higher performance, whereas key management is easier in asymmetric systems. The decision which security protocols and primitives to use mainly depends on the requirements of the application.

2.4 Conclusion

In Chapter 2 we identify three main types of security threats for RFID systems: Tracking and tracing, cloning and counterfeiting, and unauthorized access to the tag. We show, that authentication is able to address all of these potential security threats. Symmetric as well as asymmetric cryptographic primitives can be used for standardized authentication protocols. Symmetric algorithms have a higher performance than asymmetric ones, but key management is easier in security systems using asymmetric primitives. Depending on the application requirements, the application developer has to decide which algorithms to use for authentication. As standardized security algorithms offer a higher level of protection and interoperability, we propose to use the AES as symmetric primitive and ECDSA as asymmetric primitive for RFID systems. For these algorithms, hardware implementations meeting the constraints presented in Section 2.1.1 are already available. With these cryptographic primitives security protocols can be built. Based on the presented authentication methods we describe the development of various secure RFID applications in the next chapter.

3

Integration of Security into RFID Applications

The cryptographic primitives and protocols, as described in Chapter 2, are the building blocks of secure RFID applications. In this chapter we describe the design and evaluation of different RFID applications. For all applications, we have defined and implemented cryptographic services in order to provide a strong level of security. First, we present different protocols that provide authentication of both reader and tags. We propose to use the AES as cryptographic primitive for symmetric authentication. This algorithm was the first standardized security algorithm to be implemented in hardware for passive RFID tags. Secondly, we present mobile coupons as a secure application for the Near Field Communication (NFC) technology. This system defines electronic coupons stored on RFID devices and protected against unauthorized modification and cloning. Third, we describe a privacy-preserving RFID application dealing with pharmaceuticals that are protected from unauthorized access.

The chapter contains results of joint work together with Manfred Aigner, Martin Feldhofer, Stefan Mangard, Elisabeth Oswald and Johannes Wolkerstorfer. Parts of the outcomes were published at the Cryptographic Hardware and Embedded Systems conference 2004 [FDW04], at the International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing 2005 [FAD05], at the Workshop on RFID and Leight-Weight Crypto 2005 [DOF05], at the D.A.CH Mobility conference 2006 [DOF06], at the IEEE International Symposium on Ubisafe Computing 2007 [DA07], at the Workshop on Pervasive RFID/NFC Technology and Applications (PerTec07) [ADF07], and at the IEEE International Conference on RFID-Technology and Applications 2011 [Dom11].

3.1 Symmetric Authentication for RFID Systems

In 2005, the first AES hardware module feasible for passive RFID tags was published in [FWR05]. With this new feature for RFID tags, authentication protocols for RFID systems using this strong cryptographic algorithm can be designed. In this section, we describe the development and evaluation of symmetric reader and tag authentication protocols. We start with the identification of RFID applications where symmetric authentication can be useful. Afterwards, we describe four authentication protocols with different security features and evaluate their performance and practicability using the ISO-18000-3 RFID communication standard.

Key management is an important issue when using symmetric primitives, as described in Chapter 2. Therefore, such protocols are more feasible for closed-loop systems, where one central instance has control over all participants. All devices can get their keys and key updates from the central instance. An airport-luggage tracking system is an example for an RFID application, which can be implemented as a closed-loop system. Each piece of luggage is equipped with an RFID tag. All available tags are registered in a central server. In that way, automated cargo and baggage transportation is possible. Also access control to restricted areas and checks if the luggage is on the same plane as the owner can be performed. In this scenario, tag cloning and unauthorized access to the tag must be avoided. Tracking and writing or reading of the tag should is only possible for authorized readers.

Further examples where symmetric authentication can be applied are car immobilizers or transportation applications. As for transportation, unauthorized parties should be prevented from sniffing the IDs of transported goods. Thus, spying and tag cloning can be avoided. Furthermore, only authorized parties should be allowed to manipulate the stored data on the tag. Having such closed-loop applications in mind, we describe reader and tag authentication protocols, which are suitable to provide the required security services, in the following.

3.1.1 Design of Security Protocols

The security protocol uses symmetric challenge-response techniques based on encryption, which are defined in the ISO/IEC 9798-2 standard [Int99]. We explain an unilateral authentication in Section 2.3.3: The verifier (V) sends a random challenge (C) to the prover (P), which encrypts (Enc) this challenge with its secret key (K) and sends back the response. The mutual authentication protocol works similarly.

$$\begin{aligned} V \leftarrow P &: C_V \\ V \rightarrow P &: Enc(C_V, C_P)_K \\ V \leftarrow P &: Enc(C_P, C_V)_K \end{aligned}$$

Here, the verifier and the prover change their roles in the middle of the protocol. First, the verifier sends its challenge (C_V) to the prover, like in an unilateral authentication. Now, the prover generates a new random challenge (C_P), concatenates it with the challenge from the verifier and encrypts the whole message. The result of the encryption is sent to the verifier. The verifier can decrypt the message and can verify that it contains the original challenge. The other part of the message is the challenge from the prover. The verifier changes the sequence of the two challenges and encrypts it again with the secret key. The prover receives the response from the verifier, decrypts it and checks whether the two challenges are correct. If the protocol is finished successfully, both parties have proven their knowledge of the key and have therefore authenticated their identities.

In Section 2.3 we have explained three security requirements for RFID systems: location privacy to prevent tracking and tracing, proof of origin to prevent cloning and counterfeiting, and access control to prevent unauthorized tag access. All these requirements can be met by authentication protocols. Depending on the security requirements of an RFID application, reader authentication, tag authentication, and/or mutual authentication services are required. In the following, we describe four authentication protocols for meeting different security requirements.

Protocol 1: Tag Authentication with UID

The tag authenticates by proving the knowledge of a secret key. If the key has been issued by the producer of the tagged product, a proof of origin of this product is provided. In this way, cloning and counterfeiting are prevented. Tracking is still possible as the tag reveals its identity by sending its UID in every inventory process. In the following protocol, R denotes the reader, which is the verifier, and T denotes the tag, which is the prover in the protocol. The "|" denotes a concatenation of values.

$$
\begin{aligned}
R &\rightarrow T &&: \mathit{Inventory} \\
R &\leftarrow T &&: \mathit{UID} \\
R &\rightarrow T &&: \mathit{AuthRequest} \mid \mathit{UID} \mid C_R \\
R &\leftarrow T &&: \mathit{Enc}(C_R \mid C_T^\star)_K \mid C_T^\star
\end{aligned}
$$

The first two steps represent the standard inventory procedure, where the tag reveals its unique ID (UID). The reader sends an authentication request ($AuthRequest$), addressed with the UID of the tag. The request contains the challenge (C_R) for the tag. The tag encrypts the challenge using its secret key (K) and sends the response to the reader, which can then verify the result. In order to avoid chosen-plaintext attacks, this means that an attacker can control the value of C_R, the tag can "hide" the challenge by also using a random number (C_T) as input for the encryption. The use of this random number is optional, which is indicated by *.

Protocol 2: Tag Authentication with Pseudonym

Like in the previous protocol, the tag authenticates to the reader. This means that proof of origin is provided. The difference to the previous protocol is the use of a pseudonym during the inventory procedure. The tag sends a different random number each time a reader performs an inventory protocol. Thus, unauthorized readers cannot reveal the ID of the tag and are therefore not able to uniquely identify an item. The UID is only sent to authenticated readers. In this way, tracking and tracing of the tag is prevented.

$$\begin{aligned} R &\rightarrow T &&: \mathit{Inventory} \\ R &\leftarrow T &&: P_T \\ R &\rightarrow T &&: \mathit{AuthRequest} \mid P_T \mid C_R \\ R &\leftarrow T &&: \mathit{Enc}(C_R \mid UID)_K \end{aligned}$$

In this protocol the tag uses the pseudonym P_T for the inventory process. The authentication request is addressed with this pseudonym. The challenge from the reader is concatenated with the UID of the tag for encryption. An authenticated reader can check the authenticity of the tag and reveal the UID by decrypting the tag's response.

Protocol 3: Reader Authentication

In this protocol, the reader authenticates to the tag. The tag can check if a reader is authorized to access its data. In this way, access control is provided and unauthorized access to the tag's memory can be avoided. The tag takes part in the inventory process with a pseudonym (P_T). This measure provides location privacy. All further reader requests use the pseudonym as tag address. After authorization of the reader, the tag sends its UID and grants access to its memory to the reader. We suggest to encrypt further communication to prevent sniffing.

$$\begin{aligned} R &\rightarrow T &&: \mathit{Inventory} \\ R &\leftarrow T &&: P_T^{\circ} \\ R &\rightarrow T &&: \mathit{ReaderAuth} \mid P_T \mid \mathit{Enc}(P_T \mid C_R^{\star})_K \mid C_R^{\star} \\ R &\leftarrow T &&: \mathit{Enc}(UID)_K \end{aligned}$$

We denote the pseudonym of the tag by $^{\circ}$ to indicate that the tag uses a flag to show that reader authentication is required. The pseudonym is used as challenge for the reader. The reader encrypts the challenge (pseudonym) and sends an reader-authentication request (*ReaderAuth*) containing the encryption result to the tag, which can then verify the authenticity of the reader. Like in protocol 1, the reader can use a random number (C_R) and combine it with the challenge of the tag. The use of this random number is optional.

Protocol 4: Mutual Authentication

In this protocol both, reader and tag, authenticate themselves. In this way, the features from the protocols 2 and 3 are accumulated. The proposed protocol provides proof of origin as well as access control to the tag's memory. Tracking is prohibited by the use of pseudonyms.

$$\begin{aligned} R &\rightarrow T &:\ & Inventory \\ R &\leftarrow T &:\ & P_T^{\circ} \\ R &\rightarrow T &:\ & MutualAuth \mid P_T \mid Enc(P_T \mid C_R)_K \mid C_R \\ R &\leftarrow T &:\ & Enc(C_R \mid P_T \mid UID)_K \end{aligned}$$

The tag takes part in the inventory procedure with a pseudonym (P_T), indicating by a flag that reader authentication is expected. The pseudonym is used as challenge for the reader. The reader also generates a random challenge for the tag and sends a mutual-authentication request ($MutualAuth$) containing the encryption result of both challenges. The tag changes the sequence of the two challenges, concatenates it with its UID and sends the encrypted value back to the reader. If all verifications succeed, tag and reader are authenticated and the reader knows the UID of the tag. All further communication should be encrypted to provide sniffing.

3.1.2 Implementation of Authentication Protocols

In order to implement the protocols defined in Section 3.1.1 various RFID communication standards can be used. All of these standards describe, besides physical aspects, the communication protocol between RFID readers and tags. The different standards use different frequencies and timing parameters to establish the communication link. In principle, the proposed authentication protocols can be implemented with any of these RFID standards. We choose the ISO-18000-3 standard [Int04b] for this purpose. This standard uses a frequency of 13.56 MHz for establishment of an RFID communication.

The implementation of our authentication protocols has to conform to the communication protocol described in the ISO-18000-3 standard. The standard defines the structure of reader requests and tag responses. Four categories of reader requests can be differentiated: Mandatory, optional, custom, and proprietary requests. Mandatory commands have to be implemented by all tags conforming to the standard. In the ISO-18000-3 standard the *Inventory* and the *StayQuite* requests are mandatory. Optional requests are also defined

SOF	Flags	Custom	IC Mfg code	Custom request parameters	CRC16	EOF
	8 bits	8 bits	8 bits	Custom defined	16 bits	

Figure 3.1: Custom Command Format in ISO-18000-3 [Int04b]

in the standard and can optionally by provided by the tags. The user can also define custom and/or proprietary requests on her own. Custom requests must conform to the frame structure defined in the standard. Proprietary requests can also have a proprietary structure. In the following, we extend the ISO-18000-3 standard to support cryptography in order to provide strong authentication of tags and readers.

Some of the protocols, described in Section 3.1.1, require that the tag indicates that reader authentication is necessary. This can be done by setting a flag in the inventory response. The standard reserves six bits of the response flags for future use. These bits can be used for our purpose. For implementation of the required additional requests we use the structure of custom requests defined in the ISO-18000-3 standard [Int04b]. This structure is illustrated in Figure 3.1.

A custom request in ISO-18000-3 consists of eight bits for the request flags, an 8-bit custom-command code (between 0xA0 and 0xDF), an 8-bit manufacturer code (ICMFG), data of variable length, and a 16-bit CRC value. For implementation of the proposed authentication protocols, three additional requests have to be defined: Authentication Request (*AuthRequest*), Reader Authentication Request (*ReaderAuth*) and Mutual Authentication Request (*MutualAuth*). These requests are mapped to the custom-command structure by choosing a command code between 0xA0 and 0xDF and by packing all necessary information in the data field of variable length.

Another point to consider for integration of authentication protocols into the ISO-18000-3 standard is the *tag waiting time*. This value defines the time frame within a tag is supposed to respond to a reader request. Note that the time is measured from the detection of an EOF (*End of Frame*) of the reader request. In ISO-18000-3, this time frame ranges from 318.6 μs to 323.3 μs. This is a very short time especially to perform cryptographic computations. In most of the cases, these computations on the tag will exceed the tag waiting time. The AES module described in [FWR05] needs about 10 ms (@ 100 kHz) to finish its calculation, for instance. This means, the execution of an encryption or decryption cannot be performed within one request and response exchange. Therefore, we have to use a modification of the standard communication flow in our protocols.

One way to overcome the issue of too short time frames, which are given by the ISO standard, is to interleave the cryptographic protocol as we have proposed in [FDW04] and also have presented in Section 2.1.1. The basic concept on interleaved protocols is to use two reader requests instead of one. When using one reader request, the tag has to provide the response in the defined tag waiting time, which is not possible for the proposed AES hardware module. Therefore, the reader request is split into two requests: the first one sends the challenge, but does not expect an immediate response to the challenge. The second request asks for the response to the challenge after a certain time delay. Figure 3.2 demonstrates tag authentication using an interleaved protocol.

The time frames in the figure are calculated with the simulation tool *PETRA*, which we describe in Chapter 4. The reader sends an authentication

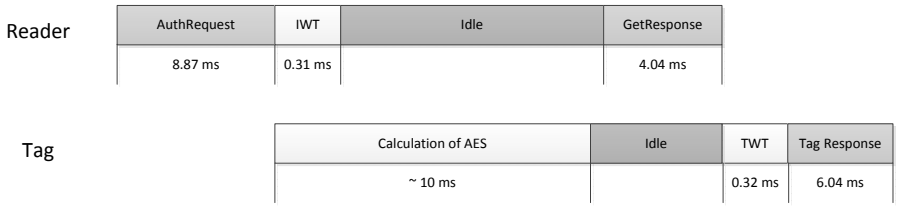


Figure 3.2: Tag Authentication with Interleaved Protocol

request and the tag starts the calculation after having received the EOF. The reader has to wait at least for 302.9 μs until sending the next request, this is called the interrogator waiting time (IWT) defined in the ISO-18000-3 standard. The calculation time of the AES is about 10 ms and exceeds the IWT by far. After a certain time, in which the reader is idle, the reader sends a request to receive the response to the challenge (*GetResponse*). If the tag has already finished its calculation, it responds to this request with the encryption result after the tag waiting time (TWT). If the tag is still busy, it does not respond and the reader has to wait again at least for the interrogator waiting time until it can issue the next *GetResponse* request.

The performance of the interleaved authentication protocol strongly depends on the AES calculation time. To increase the performance of the proposed protocols, the interleaved protocol offers the possibility of concurrent tag sessions. The reader can use its idle time to start new protocols with other tags. Figure 2.1 in Chapter 2 shows the concept on the concurrent interleaved approach involving three tags. We discuss the impact of this concurrency on the performance in the following section, where we evaluate the performance of the different authentication protocols.

3.1.3 Evaluation of the Protocols

For development and evaluation of the proposed security protocols, we have designed a simulation tool called *PETRA*. It simulates the ISO-18000-3 communication standard for RFID systems. The user implements own applications by using mandatory, optional, and custom reader requests. Furthermore, the behavior of the tag during simulation can be modified. The output of *PETRA* is a logfile containing the communication protocol flow, timing, internal behavior of reader and tags, as well as all communication frames of the protocol. Timing requirements of tags and reader can be defined by the user, too. She can, for instance, define that an AES calculation needs 10 ms. This feature is required to simulate interleaved protocols.

When programming of the host application is finished, the user defines the tags that should be simulated. The number of tags, as well as their moving behavior is set. During simulation, *PETRA* generates a log-file, where reader and tags communication and timing is logged. The software offers the possibility to consecutively execute the host application with random tag UIDs automatically

Table 3.1: Performance Evaluation of Authentication Protocol using AES

	1 tag / 1 slot	20 tags / 16 slots
Inventory	11 ms	365 ms
Protocol 1	48 ms	798 ms
Protocol 2	51 ms	847 ms
Protocol 3	51 ms	864 ms
Protocol 4	53 ms	903 ms

and to calculate the average protocol execution time. We describe this property and the simulation tool *PETRA* in detail in Section 4.2.1. We use the simulation tool for performance evaluation of the authentication protocols defined in Section 3.1.1. Table 3.1 shows some of the results of this evaluation.

The figures in the table represent the average protocol-execution time for 100 executions of one particular protocol. The AES-calculation time for the tag is set to 10 ms. Therefore, the protocols can only be performed in interleaved mode, as the AES calculation on the tag exceeds the tag waiting time. We assume that the AES calculation on the reader does not exceed the interrogator wait time. When considering only one tag in the field, we apply the anti-collision protocol with one slot describe in the ISO-18000-3 standard. Using this approach, all tags answer the inventory request at the same time. For 20 tags in the field, we apply the 16-slots variant. In this variant, the anti-collision procedure is split into 16 slots, where the slot number is represented by four bits. Tags answer in the slot where the four least significant bits of the UID correspond to the slot number.

The time required for performing an authentication protocol is about 5 times the execution time of handling a standard inventory request for one tag. The more tags are in the field the more concurrency can be exploited using the interleaved protocol mode. In this way, the time needed for executing the protocol involving 20 tags is only about 3 times longer than the time for the standard inventory protocol. With the figures gained from the simulation of the protocols we show that the performance overhead for authentication is small enough to consider these techniques in real-world applications. In the next section we discuss the feasibility of symmetric authentication using the AES algorithm for another RFID communication standard (EPCglobal Gen-2).

3.2 Feasibility of AES Authentication for EPC Gen-2

In this section, we investigate the feasibility of symmetric authentication protocols using the AES algorithm on EPCglobal Gen-2 tags [Int04c]. The EPCglobal

Gen-2 standard is an RFID communication standard for UHF tags. In this standard, tags are uniquely identified by their Electronic Product Codes (EPCs). In [Plo07], tag authentication with AES is suggested as a security enhancement for the EPCglobal Gen-2 standard. This suggestion has been adapted from our previous work described in Section 3.1. As in the previous section, we use a challenge-response authentication protocol for the feasibility study. The calculation time of the AES exceeds the tag waiting time. Therefore, we implement the authentication protocol in an interleaved mode. In the following, we describe the proposed authentication protocol and the results from the performance evaluation in detail.

3.2.1 Implementation of Authentication Protocol

In order to implement software prototypes of the tags, we use the software simulation tool *RFIDSim* ([FWS08], [FP08]). The simulation is used to verify the functionality of the tag and the protocol as well as for performance evaluation of the protocol. *RFIDSim* implements the EPCglobal Gen-2 standard. The user can define various application parameters (number of readers, number of tags, tags mobility, etc.), system parameters (data rate, timing constants, etc.), as well as physical parameters (field strength, fading, error rates, environmental temperature and noise, etc.).

For implementation of the authentication protocol using *RFIDSim*, we consider a system consisting of one reader and several non-moving tags. The EPCs and the locations of the tags are chosen randomly at the beginning of one simulation run. The error rates due to field effects and noise are set to zero during simulation. The reader *tari* value, which is defined as the reference interval for reader-to-tag signaling in the communication standard, is set to 25 μs. This value is the basis for all further timing estimations. For more detailed information on the EPGglobal Gen-2 timing refer to [Int04c].

We simulate the system with a constant slot count for all inventory rounds. The concept on the slotted inventory process is defined in the EPCglobal Gen-2 standard [Int04c] and works as follows: The tags choose a random number in the range of the number of slots at the beginning of the inventory process. This random number is the slot count for the tag in this particular inventory process. The reader starts the inventory process by asking all tags with the slot count 0 to answer. Afterwards, the slot count of the reader is incremented and all tags with a slot count of 1 answer the next request. This is repeated until all slot counts have passed through.

Figure 3.3 illustrates the tag-state diagram for Gen-2 tags. It contains the inventory procedure up to the commands to get the tag into the *Open State*. Tags have to reach the *Open State* in order to handle further custom protocols. At this point, the handling of the proposed authentication protocol starts.

For evaluation of the symmetric authentication protocol using the AES algorithm we implement a challenge-response protocol, as described in Section 2.3.4. We use the AES hardware module presented in [FWR05], which has an AES calculation time of about 10 ms. The AES calculation time exceeds the tag

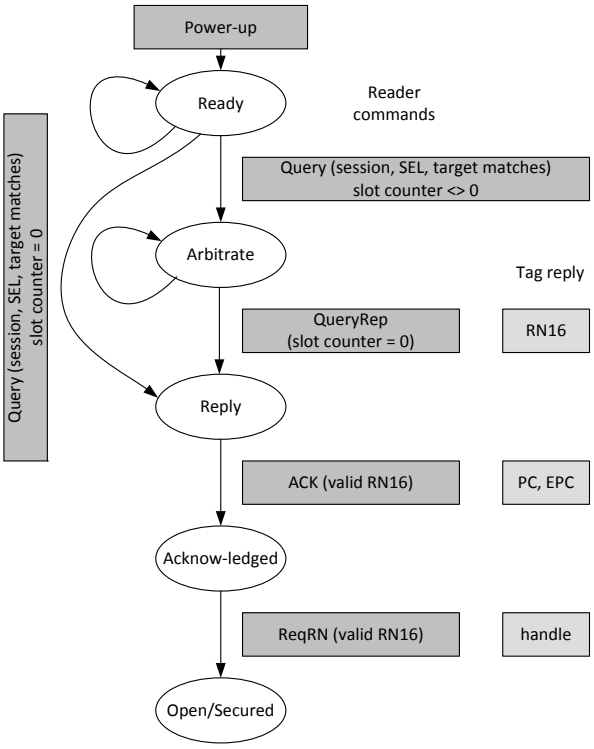


Figure 3.3: Tag State Diagram for EPCglobal Gen-2 Tags

response time of 187.5 μs. This means that the authentication protocol has to be handled as interleaved protocol (see Section 2.1.1). Two separate requests are required to perform the tag authentication: One for sending the challenge, and one for getting the response. These two commands can be integrated into the EPCglobal Gen-2 standard by defining them as custom commands. They can be issued by the reader when the tag is in *Open State*. The structure of the requests and the structure of the tag responses are defined in [Plo07]. The principle of the interleaved authentication protocol, used for performance evaluation, is shown in the following.

$$\begin{aligned} Reader &\rightarrow Tag &: \quad & SendChallenge(C) \\ Tag &\rightarrow Reader &: \quad & ChallengeReceived \\ Reader &\rightarrow Tag &: \quad & GetResponse \\ Tag &\rightarrow Reader &: \quad & Enc(C)_{Key} \end{aligned}$$

We compare two different strategies as regards the protocol sequence: The two authentication commands (*SendChallenge*, *GetResponse*) can either be sent within one inventory slot or can be separated and issued in different inventory rounds. One inventory round has 2^Q slots. The value of Q can be defined by

the user. Following the first strategy, the two authentication commands are issued consecutively during one single inventory slot. Right after identifying a tag, the reader sends the corresponding commands to get the tag in *Open State*. Then a *SendChallenge* command is issued. The tag answers with its handle, which is a number to identify one tag during one particular session. Directly after receiving the tag response, the reader issues a *GetResponse* command. If the reader does not receive any response, it polls the tag again until it reaches a time-out or the tag sends a response. The maximum number of *GetResponse* commands before time-out is defined in the reader application. After finishing the inventory round, *Select* commands are issued for each of the tags to exclude them from further inventory procedures. We refer to this method as *one-slot approach*.

Following the second strategy, the two commands are separated and issued in different inventory rounds. After the reader has identified a tag, it gets the tag in *Open State*. A *SendChallenge* command follows. The reader ends this part of the authentication session and continues with the inventory procedure. The reader stores the challenges it has sent to the tags together with their EPCs in a database. If the reader identifies one of the tags in the database in one of the following inventory rounds, it gets the tag again in *Open State* and sends a *GetResponse* command. If the tag's response corresponds to the encrypted challenge, the tag is authenticated. We refer to this strategy as *different-round approach*. In the next section, we present a performance evaluation of both strategies when using different numbers of tags and slots, and with different amount of time required for the AES calculation.

3.2.2 Performance Evaluation of Authentication Protocol

The performance of an authentication protocol, using the AES algorithm, implemented with the EPCglobal Gen-2 standard depends on the numbers of tags in the field, on the slots used for one inventory round, and on the calculation time for the AES algorithm. The authentication protocol is implemented as an interleaved challenge-response protocol (as described in Section 3.2.1). This means that two commands are used: One for sending the challenge to the tag, and one for requesting the response from the tag.

For simulation, we have defined AES calculation times of 0 ms up to 24 ms in 3-ms steps. An AES calculation time of 0 ms does not mean that the AES algorithm requires 0 ms, but that the calculation time is smaller than the tag waiting time. In Table 3.2, the results for the simulations with an AES calculation time of 0 ms, 9 ms, and 24 ms are shown. The column `Ref.` lists the time required to perform a reference protocol, which is a protocol consisting of an inventory procedure until getting the identified tags into *Open State*. The columns named `OS` represent the *one-slot approach*, which we have described in Section 3.2.1. The columns named `DR` represent the *different-round approach*, also described in the previous section. The figures in the table represent the average protocol-execution time over 50 protocol runs measured in milliseconds.

The table shows that the performance strongly depends on the combination

Table 3.2: Selected Performance Figures for Gen-2 Authentication Protocol

			AES Calculation Time					
			0 ms		9 ms		24 ms	
Tags	Slots	Ref. [ms]	OS [ms]	DR [ms]	OS [ms]	DR [ms]	OS [ms]	DR [ms]
1	1	13.2	34.3	43.8	43.8	54.9	59.9	68.9
1	2	13.9	35.0	45.2	44.5	54.9	60.7	69.8
1	4	15.1	36.2	47.7	45.9	61.0	60.9	74.1
1	8	17.6	38.8	52.7	48.3	64.3	64.6	79.0
5	2	76.3	182.0	260.0	231.2	290.6	318.5	329.6
5	4	65.0	171.1	228.2	220.1	252.7	298.7	273.4
5	8	65.7	170.0	229.4	220.2	240.6	301.6	257.4
5	16	69.9	176.7	240.3	225.3	256.3	308.4	283.6
10	4	142.1	354.6	484.4	451.0	537.2	618.0	593.9
10	8	129.4	342.5	474.0	438.3	475.5	608.0	511.7
10	16	135.1	344.3	462.2	442.6	501.0	617.9	524.0
10	32	147.3	362.8	489.8	460.1	532.3	632.5	573.5
50	16	726.3	1,786.3	2,421.0	2,282.5	2,551.4	3,129.2	2,653.1
50	32	651.8	1,708.8	2,345.0	2,215.5	2,438.3	3,058.3	2,509.2
50	64	671.4	1,723.4	2,420.9	2,256.0	2,473.9	3,105.8	2,611.9
50	128	732.9	1,780.5	2,572.1	2,346.6	2,675.5	3,252.8	2,842.7
100	128	1,355.8	3,454.9	4,921.9	4,554.1	5.017,7	6,292.0	5,198.6

of tags and slots and on the AES calculation time. If the number of slots is similar to the number of tags, the performance increases. As expected, the timing overhead raises as the AES calculation time gets longer. The overhead factor for authentication ranges from 2.20 to 5.23 in respect to the reference protocol. We demonstrate the results and its dependence on the AES calculation time in Figure 3.4.

The four graphs show the performance figures for different tag and slot numbers. The graph for one tag and one slot illustrates, that the overhead for the *different-round approach* is higher than for the *one-slot approach* for this setting. This is logical, because for one tag no pipelining of the protocol execution can be reached. For all other combinations of tags and slots, the *different-slot approach* is worse than the *one-slot approach* up to a certain AES calculation time. For 50 tags and 64 slots, the intersection point lies between 13 ms and 14 ms of AES calculation time. If the AES calculation time is longer, the *different-slot approach* becomes favorable in terms of performance. The more tags are in the field the earlier this point is reached. Then, the *different-round approach* has an advantage over the *one-slot strategy* because the protocol execution can be pipelined.

During the feasibility study we implemented software prototypes of EPC-global Gen-2 tags providing symmetric authentication. The simulation tool *RFIDSim* was used to implement and verify the functionality on the tag. The protocol was simulated with different parameters (number of tags, number of slots, AES calculation time) to get an idea of the practical impact of using authentication techniques on the performance. Furthermore, we have shown the

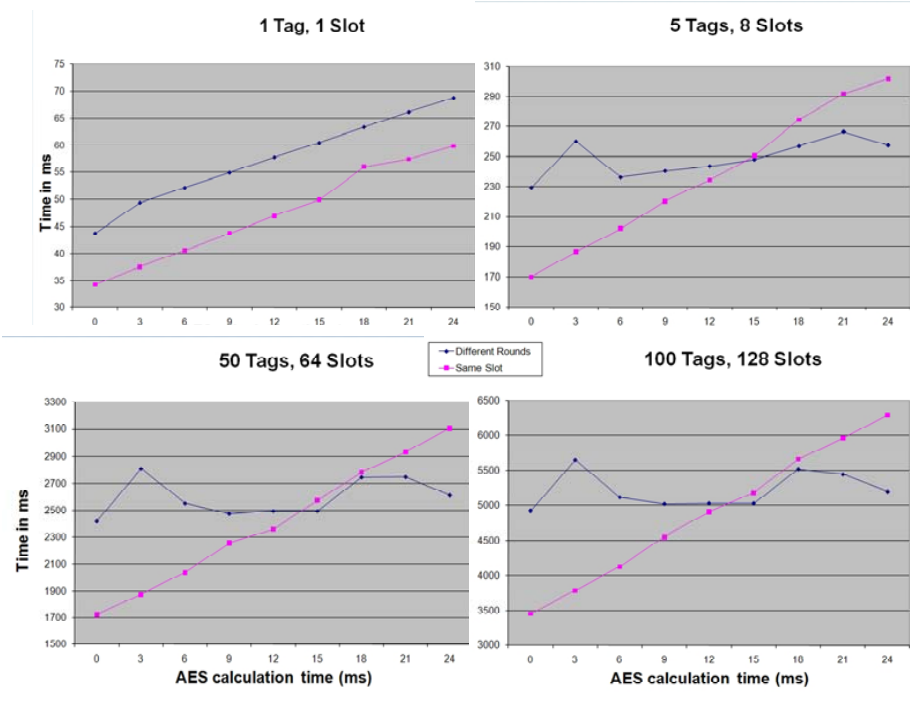


Figure 3.4: Gen-2 Authentication Depending on AES Calculation Time

impact of different protocol-execution strategies.

Authentication can be used as countermeasure to various security threats in RFID systems. We have discussed this fact in Section 2.3.4. In this section, as well as in Section 3.1, we have dealt with the feasibility of symmetric authentication for different RFID communication standards. In the next section we describe the development of an RFID application where symmetric as well as asymmetric primitives are used to prevent cloning, counterfeiting and unauthorized tag access.

3.3 Secure Mobile Coupons

In this section, we describe a system of electronic coupons that can be transmitted from an RFID tag to an RFID reader device. Passive RFID tags, which are, for instance, attached to a billboard or magazine, act as issuers of the coupons. The RFID readers are the clients that collect the coupons from the tags and carry them to another RFID reader, the cashier. Here, the mobile coupons (mCoupons) are cashed in.

As mCoupons have a certain value, security is an important requirement for the developed system. Unprotected mCoupons can be cloned and modified. The

mCoupon protocols use symmetric and asymmetric cryptographic primitives to provide the following security features:

- An attacker is not able to use the same mCoupon multiple times (multiple cash-in).
- An attacker is not able to generate new valid mCoupons (counterfeiting).
- An attacker is not able to change the information of an mCoupon without invalidating it (unauthorized access).
- An attacker is not able to produce a valid copy of an mCoupon (cloning).

The mCoupons system has been developed on basis of the *Near Field Communication* standard (NFC) [Int04a]. NFC is an RFID communication standard that works on a frequency of 13.56 MHz. The applications of NFC follow the principle of touching communicating devices. This means that devices are brought closely together (a few centimeters) to share data. NFC provides the feature that NFC devices can act in both, active and passive mode. In passive mode, the initiator of the communication establishes a radio-frequency field that is used from the passive participant to send data over the air interface. In active mode, both communication devices generate their own radio-frequency field for data transfer. In the following, we explain why these two modes are the reason to select the NFC standard for implementation of the mCoupon protocols.

The clients in the mCoupon system act on the one hand as RFID readers, when collecting the mCoupons. On the other hand, the clients are supposed to communicate with other RFID readers, when cashing in the mCoupons. The communication between two RFID readers over the radio link is in general not provided by an RFID communication standard. Nevertheless, using the NFC standard, two active devices can communicate with each other. We use the active mode for communication between client and cashier. For the link between the passive issuer tag and the client we use the passive mode of NFC.

3.3.1 The mCoupon Protocol

The following parties are involved in the mCoupon protocol: The issuer, the client, and the cashier. An mCoupon is issued by the issuer, which is a passive RFID device attached, for instance, to a newspaper advertisement or a poster. The client has to "touch" the issuer to establish a connection and receive the mCoupon . The client takes the mCoupon to a cashier that verifies the validity. If verification is successful, the cashier hands the value represented by the mCoupon (a product or service) over to the client.

Two protocols providing different security services have been developed for the mCoupon system: The "simple" protocol provides protection against counterfeiting and modification of the coupons. The "advanced" protocol extends the simple version by cloning and copy protection. In the following, we describe both protocols in detail.

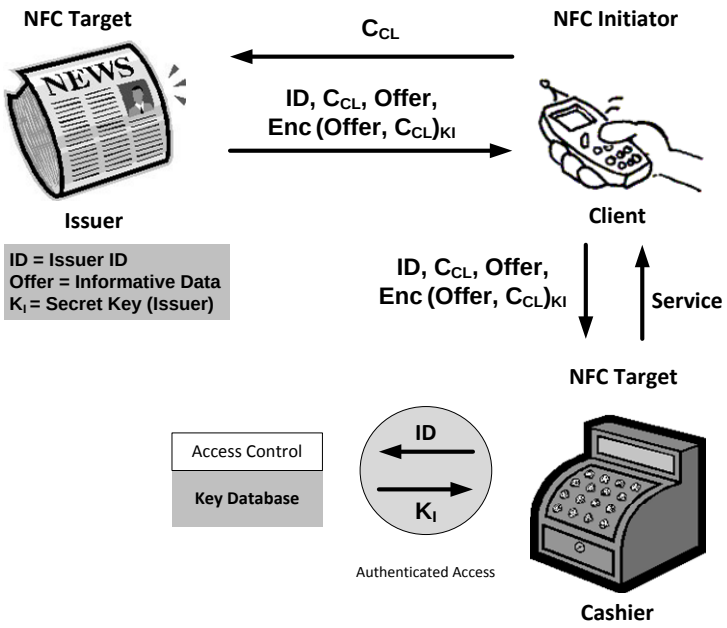


Figure 3.5: Simple mCoupon Protocol

The Simple mCoupon Protocol

The simple mCoupon concept works with symmetric cryptography. We suggest to use the AES algorithm as cryptographic primitive. Each issuer holds a secret key, which is also stored in a key database. Only authorized cashiers have access to this database. Figure 3.5 shows the issuing and the cashing-in of an mCoupon in the simple protocol variant.

The client wants to receive an mCoupon from the issuer and initiates a connection by sending a random number as a challenge (C_{CL}). The issuer encrypts this random number and the offer with its secret key (K_I). The offer basically represents information on the mCoupon. Together with the encryption result, the issuer sends its ID, the random number, and the offer to the client. The client stores this response, which represents the mCoupon, and carries the mCoupon to a cashier. The client connects to the cashier via radio link and sends the mCoupon. In order to verify the validity of the coupon, the cashier connects to the database and sends the issuer's ID. The database verifies the authenticity of the cashier and sends the secret key corresponding to the ID to the cashier. This transfer is secured to provide confidentiality. Authentication of the cashier to the database is required for the proposed protocol. We do not define the authentication mechanism at this point, as authentication mechanism between two computers on the Internet are already well established. We suggest to apply a standardized authentication procedure for network connections.

Counterfeiting and modification of the mCoupon is prevented, as only autho-

rized issuers and cashiers know the secret key. Multiple cash-in can be prevented on the cashier's site. Each mCoupon has to hold a unique identifier (e.g., the ID of the issuer and an ascending number). The cashier can, for instance, store the identifiers of mCoupons that are already cashed-in in the central database. If a new mCoupon is transmitted to the cashier, it can check if the mCoupon of the client is already stored in the database. In this way, the cashier can decide if the coupon has already been cashed-in before.

Cloning and Copying of the mCoupons from one client to another is possible in the simple mCoupons protocol. An attacker can sniff the conversation and store the mCoupon (plain and encrypted information) on the tag. The mCoupon could also be transferred to another device. In the following, we describe an advanced protocol which extends the features of the simple protocol by copying and cloning protection.

The Advanced mCoupon Protocol

The advanced protocol involves, besides symmetric algorithms, asymmetric cryptographic primitives to provide additional authentication of the client. With this measure, cloning of the mCoupon is prevented. The client generates a key pair consisting of public and private key which are stored in a secure memory. The client transfers its ID and the public key to a *Public Key Infrastructure* (PKI) server. We have described the principles of a PKI in Section 2.3.4. The client can also get a certificate from a *Certification Authority* (CA). In both approaches, either using a PKI server or certificates, the cashier can get the public key of the client and authenticate it. This feature is used to provide cloning and copy protection. In the following we will describe the advanced mCoupon protocol, illustrated in Figure 3.6, in detail.

As in the simple protocol, the client "touches" the issuer and sends a request to receive an mCoupon. This request contains a random number (C_{CL}). In the advanced protocol, the issuer wants the client to authenticate and sends as well a challenge (C_I) to the client. The client signs this challenge by using its private key ($PrivK_{CL}$). The client sends its ID (ID_{CL}) and the signature to the issuer. The issuer is not able to verify the signature, because of its limited computational resources. Therefore, the verification of the signature is postponed until the mCoupon is cashed in. The issuer encrypts the following values: The client's ID, the challenge C_I, the signature received from the client, the offer, and the random number C_{CL}. Now the issuer generates the mCoupon, which consists of the issuer's ID, the challenge C_{CL}, the offer, and the encryption result, and sends it to the client.

At cashing-in, the client transfers the mCoupon data to the cashier. The cashier wants the client to authenticate and sends a challenge (C_{CA}). The client signs this challenge and sends it together with its ID to the cashier. The cashier has meanwhile encrypted the mCoupon and can compare the identity from the coupon and the received identity. The public key of the client can either be looked up through an online PKI service or can be derived from a certificate. Using a PKI service, the cashier has to trust this service. Using certificates,

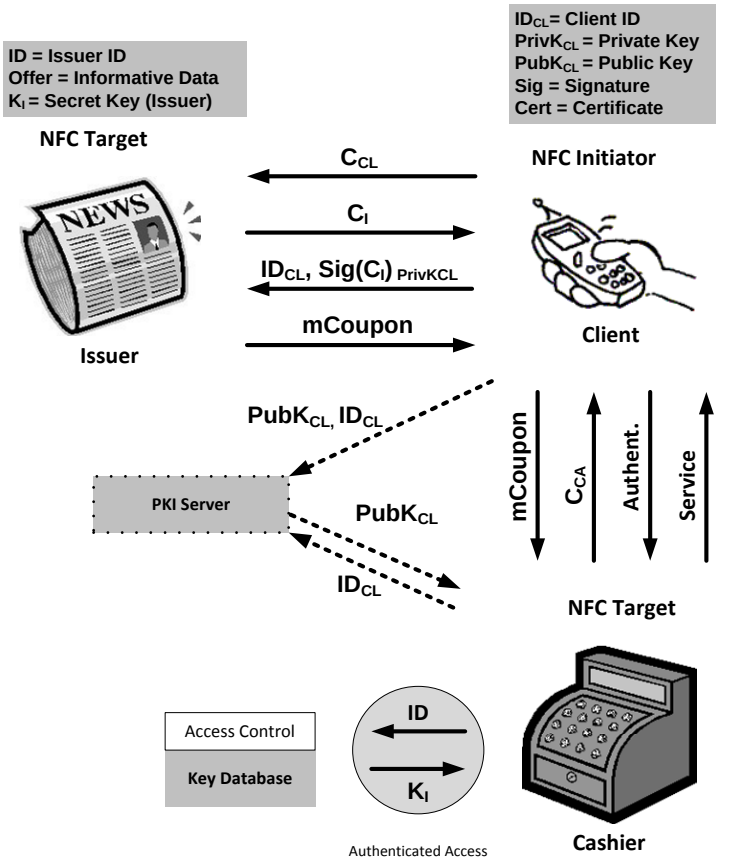


Figure 3.6: Advanced mCoupon Protocol

the client has to send the certificate, signed by the CA, to the cashier. The certificate can be verified by the cashier, if he knows the CA's public key. Once, the cashier has received a valid public key from the client, it can verify the signature over C_{CA} and also the signature contained in the mCoupon. If both signatures are valid, the cashier can be sure, that the mCoupon has been issued to the same client that wants to cash it in.

Like the simple protocol, the advanced protocol offers countermeasures against unauthorized access and counterfeiting of mCoupons. Furthermore, cloning and copying is prevented, as the client has to authenticate itself during issuing as well as during cashing-in. Only if the same public key can be used to verify both authentication results and if the client's ID is linked to that public key, the mCoupon is valid.

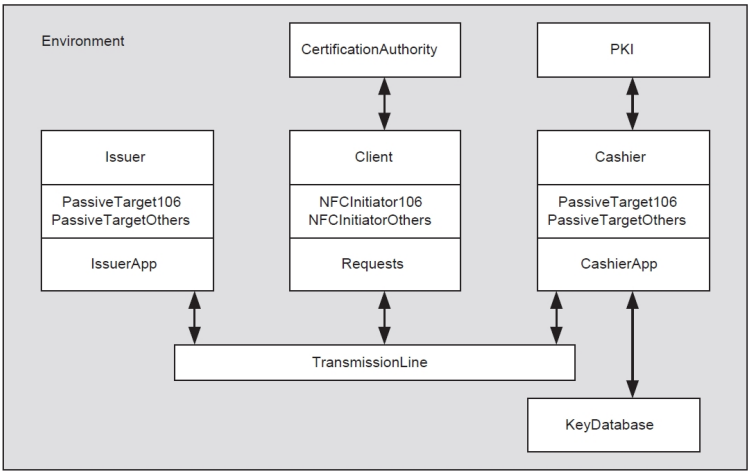


Figure 3.7: mCoupon Protocol Simulation

3.3.2 Simulation of the Protocols

For designing and evaluating the protocol we have developed an mCoupon-protocol simulator in Java. In Figure 3.7 the structure of the mCoupon software is shown. The simulator is able to emulate the behavior of the mCoupon devices conforming to the NFC standard [Int04a]. The software supports the definition of real-world protocol parameters such as command codes or datarates.

The output of the mCoupon simulator is a log-file containing all sent and received frames conforming to the chosen datarate in the NFC standard. With this log-file, verification and modification of the protocol functionality is supported. The Java simulator can also be used for the generation of reference values that can be used as test vectors for further implementations of the protocols.

In this section we have described the development of a system of mobile electronic coupons. We provide security measures that prevent counterfeiting, cloning, and copying of mCoupons, as well as unauthorized access to the mCoupons. In the next section we present an application where privacy is a strong requirement.

3.4 Preserving Privacy in RFID Applications

In this section, we deal with privacy for RFID systems. We define a set of privacy-preserving mechanisms to prevent tracking and tracing and also unwanted memory access. In order to demonstrate how these mechanisms work, we have developed a demo application (MedAssist) published in [Dom11]. For design of the application and implementation of the application prototype we have designed and implemented the simulation and prototyping tool *ProtEx*. In

Section 4.2.2 we describe this tool in detail.

One focus during the development of the MedAssist was the integration of RFID devices into the *Internet of Things*. Remote authentication of a network device to a tag can be a useful feature for the MedAssist application, as well as for many other secure RFID applications. In order to enable remote access to a tag, a concept on two-way communication with passive RFID tags using the Internet Protocol version 6 (IPv6) has been developed. Furthermore, secure connections on top of this concept has been investigated. The secure connection of passive RFID tags to the *Internet of Things*will be discussed in detail in Part II. In the following, we describe the MedAssist application. We discuss security requirements and the developed security protocols, which provide location and data privacy for the application.

3.4.1 The MedAssist Application

Privacy can be defined as the right of the user to control the access to his data. This is especially important when dealing with sensitive data. For demonstration purposes, we design a demo application where medical information is protected from unwanted access. The threat scenario is obvious, as medical data are in general treated as sensitive ones. The application is also supposed to offer an added value to the consumer. In the proposed application, the added value is offered by a device called *Medical Assistant* (MedAssist). The *Medical Assistant* can be a chest to store pharmaceuticals and/or can have sensors to observe the physical status of its owner (e.g., blood pressure or glucose level) and is able to:

- Suggest pharmaceuticals for particular diseases,
- give information about side effects of pharmaceuticals,
- show contraindications of a group of pharmaceuticals or with personal properties of the owner (e.g., hypertension, diabetes),
- alert if pharmaceuticals are expired,
- alert the consumer of taking his medicine, and
- log the usage of pharmaceuticals.

Pharmaceuticals in the proposed application are tagged during production. The producer delivers the pharmaceutical over the supply chain to the retailer (pharmacist). In the supply chain and in the retailer's shop, the tag information can be read by anyone with a standard RFID reader. At the moment a consumer buys the product, the tag is personalized for this consumer. This means that from the point of sales (POS) the tag is exclusively under the control of the consumer. The user wants the *Medical Assistant* to communicate with the tags on the pharmaceuticals, but she wants the tags to exclude all other readers from accessing the tag's data. An employer, for example, could use information

about medication to the disadvantage of an employee. Therefore, the consumer can grant access rights to particular readers and exclude others from tag access. Figure 3.8 illustrates the basic application steps. In the following, we describe the parties involved in the application protocol.

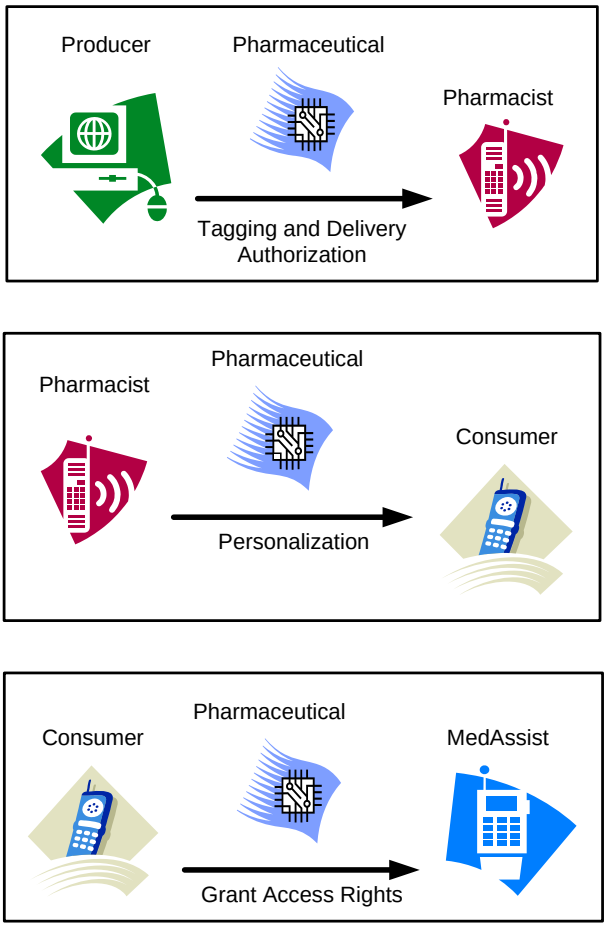


Figure 3.8: MedAssist Application

Producer

The producer adds tags to her pharmaceuticals. These tags can be used for logistic purposes and are freely readable for all readers until a certain point in the supply chain. When reaching this point, the tag is personalized and is further controlled by only one owner. The producer can decide to equip the tag with a key pair consisting of a public and a private key to enable a proof of origin of the tag. For this purpose, the producer has to sign the public key

and store the certificate on the tag. Furthermore, the producer has to assign authorization rights to dedicated retailers.

Retailer

The retailer (pharmacist) can use the tags to ease inventory management in her store. All readers can access the tags at this point. Everybody holding a reader can, for example, perform a proof of origin. If a customer buys a pharmaceutical, the pharmacist initializes the personalization of the tag. The personalization has to be done by a trusted party, this means either by the producer or an authorized retailer. Authorization of the retailer has to be done by the producer. When the tag is personalized, only the owner can control the access to the tag.

Consumer

The consumer holds an RFID reader (e.g., a mobile phone with NFC capabilities). This device is defined as the manager of the tag at personalization. Using this device, the consumer is able to manage access rights to the tag. For this purpose, a graphical user interface is available on the managing device. The managing device stores the keys and IDs of applications, the consumer wants to authorize. Authorization of an application can grant full access, but can also be restricted access to particular memory areas and/or particular commands.

Application

The application, in our case the *Medical Assistant*, has an integrated RFID reader. The NFC device of the consumer can read out the identity and key of the *Medical Assistant* and stores this information on the managing device. The consumer grants access rights to the application by configuration of the tag. The tag is configured by use of the managing device. The key of the application is stored together with the defined access right in the *Access Rights Database* of the tag. Depending on the information the application is able to collect, different services can be provided to the consumer.

3.4.2 The Security Protocol

In this section, we present the communication protocols for the MedAssist application. Security measures are integrated into the protocols in a way that location and data privacy as well as transfer of ownership are provided. The security protocol uses the Elliptic Curve Digital Signature Algorithm (ECDSA) [Nat00] as cryptographic primitive. In the following, we describe the protocol steps required for application execution in detail.

Issuing of the Tag

During the issuing process, the producer stores her public key as master key on the tag. Furthermore, each tag can optionally be equipped with an own private/public key pair for tag authentication. The producer signs the public key of the tag and stores the resulting certificate on the tag. With this certificate, the tag can perform a proof of origin. Figure 3.9 demonstrates the issuing procedure.

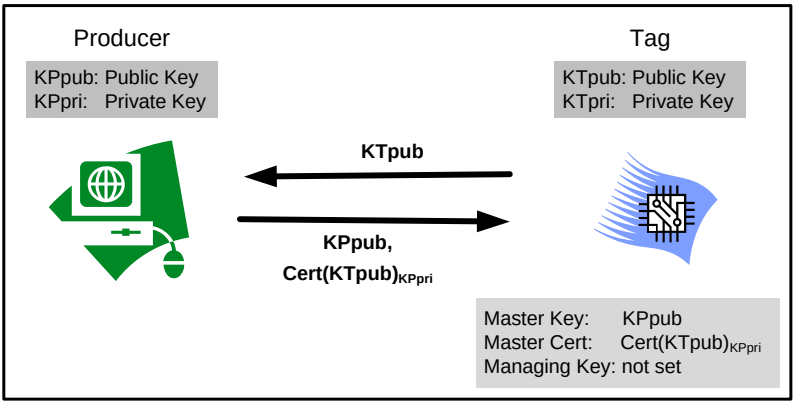


Figure 3.9: MedAssist - Issuing of the Tag

Authorization of the Retailer

Only authorized retailers can personalize the tag. Retailers are authorized by the producer. The retailer generates a key pair consisting of a public and a private key. The retailer sends her public key as well as some prove of authentication (e.g., a trading license) to the producer. The producer signs the public key and sends the resulting certificate back to the retailer. Figure 3.10 shows the issuing of a certificate for the pharmacist.

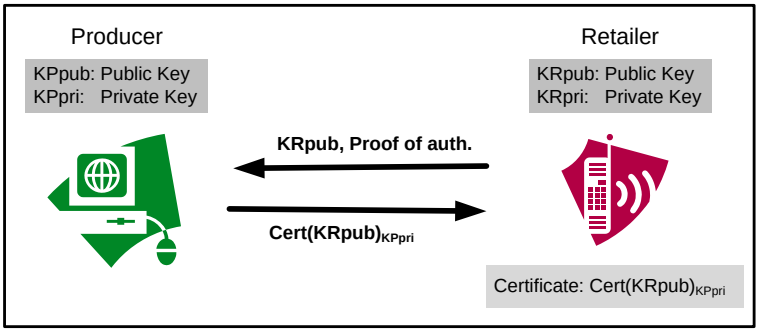


Figure 3.10: MedAssist - Authorization of the Retailer

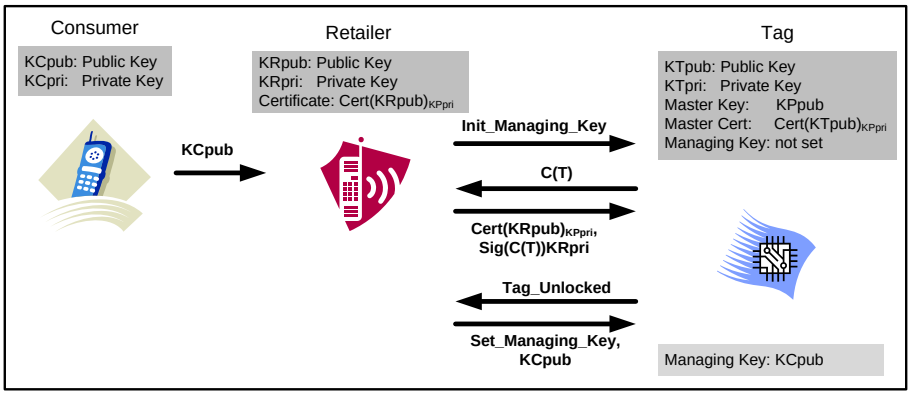


Figure 3.11: MedAssist - Personalization of the Tag

Personalization of the Tag

Personalization can only be done by authorized parties, which are the producer or authorized retailers. In general, the retailer performs the personalization for her customers. The customer possesses a managing device holding a public/private key pair. At the POS, the public key of the managing device is transferred to the RFID reader of the retailer. The retailer sends a request to perform the initial personalization to the tag. The tag answers with a challenge for the retailer, which should now prove her authorization. The retailer signs this challenge with her public key using the ECDSA algorithm. The retailer sends the signature together with her certificate to the tag. The tag can verify the certificate of the retailer with the public key of the producer, which has been stored as master key during the issuing process. If the certificate is valid, the tag verifies the signature of the challenge. If this proof is successful, the retailer is authenticated and the tag accepts the transfer of the new managing key. Figure 3.11 illustrates the personalization of the tag.

Another variant of personalization is an online procedure. This process is necessary if the producer wants to perform personalization herself and does not delegate this task to the retailers. In this case, the retailer has to connect to the producer's server and log in via a challenge-response protocol using the private key. The first part of the protocol works as in the previous described protocol until the tag sends a challenge to the retailer. Now, the retailer forwards the challenge of the tag to the producer. If the retailer is authenticated, the producer signs the challenge with her public key and sends the result to the retailer, which once again forwards this response to the tag. The tag can verify the response by using the stored master key. With this online approach the revocation issue can be addressed. If a retailer loses the authorization, the producer does no longer sign the tag's challenge on behalf of the retailer. For the offline approach, using certificates, revocation is a problem as the tag has no possibility to check the current validity of the retailer's certificate.

Access Control

The customer, as owner of the tag, has the exclusive authority to define access rights of other readers or applications on the tag. For this purpose, the customer holds a managing device. The public key of the managing device is stored as managing key on the tag. From the point of personalization, all read or write activities on the tag are denied for readers other than the managing device. The customer can send a request to grant access rights for particular applications to the tag. The managing device holds the public keys of these applications. The identity and the public key of the application as well as a list of commands, which the application is allowed to perform, is sent to the tag. The (*Set_Access_Rights*) command has to be authenticated by the managing device. The tag sends a challenge and the managing device signs this challenge with its private key. The response is sent back to the tag. If verification is successful, the tag stores the new access data in the *Access Rights Database*. Also the ID of the tag is only revealed to authorized readers. The inventory procedure is performed with a pseudonym in order to provide location privacy. Figure 3.12 shows the definition of new access rights.

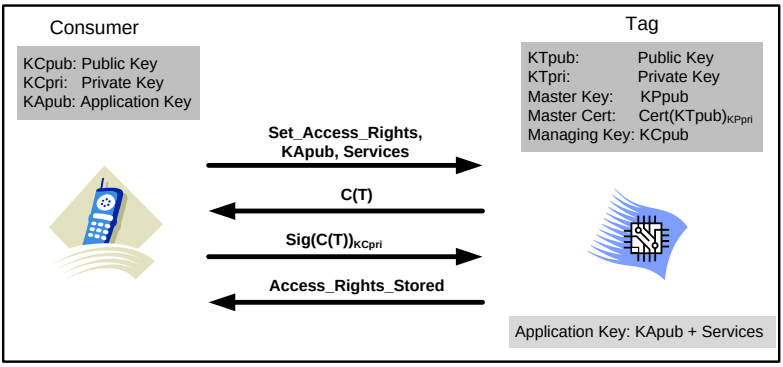


Figure 3.12: Setting of New Access Rights

Accessing the Tag

All readers (applications, respectively) that are permitted to access certain tag services can request access to the tag. The reader starts this procedure by sending its identity, public key, and a (*Service Request*). The *Service Request* contains the command that the reader wants the tag to process. If the public key is stored as an *application key* in the *Access Rights Database*, the tag indicates that authentication is required and sends a challenge. The application owns a public/private key pair and signs the challenge with its private key. If the signature is successfully verified by the tag, the tag sends the response to the processed *Service Request* to the reader. Figure 3.13 illustrates the tag access.

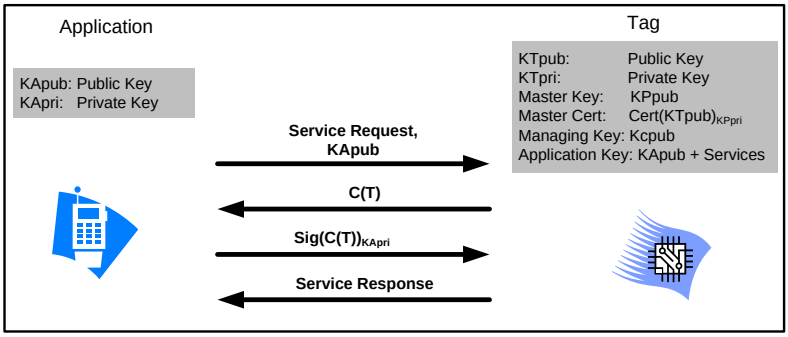


Figure 3.13: Access to the Tag

Transfer of Ownership

The customer can decide to transfer the control over the tag access to another user. The user becomes the new manager of the tag. For this purpose, the public key of the managing device, belonging to the new owner, is transferred to the tag. This command (*Set New Owner*) can only be performed by the current tag manager. The tag requests authorization and sends a challenge. The previous owner (= customer) signs this challenge and sends the response to the tag. If verification is successful, the tag stores the public key of the new owner as managing key and clears the *Access Rights Database*. Figure 3.14 demonstrates the transfer of ownership.

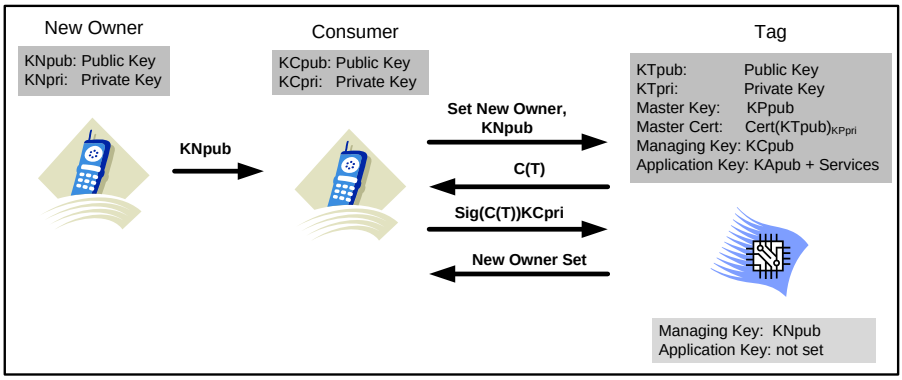


Figure 3.14: Transfer of Ownership

Recovery

If the customer somehow loses access to her private key, she should be able to recover ownership of the tag. Therefore, the public key of the producer is still stored as master key on the tag. The only service that is allowed with this

key at this point is to set a new managing key. The customer can contact an authorized retailer to set a new managing key to the tag. The recovery works like the personalization shown in Figure 3.11. The recovery should only be performed, if the customer can authenticate itself (e.g., by showing the bill) as the legal owner of the product.

For recovery, an online scenario is possible, too: The producer personalizes the tag over the Internet. As no retailer is involved in this procedure, the customer has to send its public key to the producer. Together with sending the public key the customer has to authenticate the ownership of the product, for example, by a code on the bill that has been communicated to the producer at purchase. If the producer has authenticated the ownership and the public key of the customer, it can establish a secure connection to the RFID tag on the product and send the new managing key. The tag authenticates this command by performing a challenge-response protocol with the producer. For this online procedure, two-way communication with passive tags on the Internet has to be provided. The online scenario is out of scope of this section, but we present an approach to connect passive RFID tags securely to the Internet via IPv6 in Part II.

3.4.3 Prototyping of the MedAssist Application

In the prototyping phase, we provide software and hardware models of readers and tags performing the MedAssist application as a proof of concept. The ISO-18000-3 standard [Int04b] is used as RFID communication standard. The first step is the development of a functional simulation of reader and tags. The simulation and prototyping tool *ProtEx* is used to adapt the tag functionality to the requirements of the application and to implement a reader application. The basic software tag in *ProtEx* already provides the handling of the basic requests in ISO-18000-3 (mandatory and optional commands). The user has to specify extended functionality (handling of custom commands) in an own Java class, called *Custom Tag*. During programming the software model, various modifications to the application specification and protocol had to be done in order to optimize the application protocol. We describe the development of RFID applications with *ProtEx* in Section 4.2.2. The outcome from simulation is a functional software model as well as the tag specification, describing the additional functionality and memory structures the tag has to provide. In the following we discuss the tag specification of the MedAssist tag in detail.

Tag Specification

The tag in the MedAssist application can have the following states: `NOT ISSUED`, `NOT PERSONALIZED`, or `PERSONALIZED`. The following commands can always be performed without authentication: *Inventory Procedure, Select, Stay Quiet*, and *Reset to Ready*.

When the tag is issued, a unique ID is assigned, the state is set to `NOT ISSUED` and a public/private key pair is generated. The tag handles all requests

like a regular tag without access restrictions. Commands that require an authentication are prohibited. The producer has to set a master key, then the tag is set to `NOT-PERSONALIZED` state. Also, a certificate can be stored on the tag. For this purpose, the public key is read out by a *Get-Public-Key* request, which is then signed with the producer's private key. The certificate is sent to the tag with a *Set-Certificate* request and has to be authenticated with the master key.

If the tag is in `NOT-PERSONALIZED` state, the tag is still accessible for all readers without authentication. Commands that require authentication are prohibited, except the *Set-Certificate* and *Set-Managing-Key* requests, which have to be authenticated with the master key. If the managing key is set, the tag gets into `PERSONALIZED` state. In this state, all commands except inventory commands have to be authenticated and are only accessible for authorized readers. The inventory procedure is done with a pseudonym to provide location privacy. The ID of the tag can be received by sending a *Get-UID* request, which requires authentication of the reader. The managing key can authorize all commands except the setting of the managing key and the certificate. The owner of a tag can use a *Set-New-Owner* request to perform a transfer of ownership and must authorize this request with the managing key. Figure 3.15 gives an overview of possible and prohibited requests in the different states of the tag.

All commands that require authentication are handled in the following way:

- A *Service Request* is issued by the reader. The tag checks whether authorization is required. If this is the case, the tag sends an *Error Response* indicating that authentication is required to access this service.
- The reader sends a *Get-Challenge* request containing its public key. The tag checks if this public key has the access rights for the requested service. If the public key is authorized, the tag returns a challenge. If the public key is not accepted, the tag returns an error response (*Access denied*). If the reader sends a *Get-Challenge* request containing a certificate (which is possible only for the *Set-Managing-Key* service), the tag returns a challenge and starts to verify the certificate. If the verification fails, the tag responds to the next request with an error response (*Access denied*).
- The reader appends the *Service Request* to the challenge and signs the result with its private key using the ECDSA algorithm. It sends an *Auth-Response* request containing the result. The tag sends a *No-Error* response to indicate that the request was regularly received. The tag verifies the signature of the challenge appended with the *Service Request* and authorizes the service if the verification was successful.
- The reader sends a *Get-Authorized-Service* request and receives the response to the previous sent *Service Request*. If verification of the signature has not been successful, the tag sends an error response (*Access denied*).

The reason for the splitting the last two steps is that the tag requires some time to perform the ECDSA calculation and certificate verification. Therefore, it is

	Not issued	Issued/ Not personalized	Personalized
Master Key	Not set	*Set Certificate* *Set Managing Key*	*Set Managing Key*
Managing Key	Not set	Not set	All commands except: *Set Certificate* *Set Managing Key*
Application Keys	Not set	Not set	All commands specified in access-rights list except: *Set Certificate* *Set Managing Key* *Set Access Rights* *Remove Access Rights* *Remove Application Key* *Clear Application Keys* *Set New Owner*
Without Authorization	All commands except: *Set Certificate* *Set Managing Key* *Set New Owner* *Set Access Rights* *Remove Access Rights* *Remove Application Key* *Clear Application Keys*	All commands except: *Set Certificate* *Set Managing Key* *Set New Owner* *Set Access Rights* *Remove Access Rights* *Remove Application Key* *Clear Application Keys*	*Inventory (random UID)* *Stay Quiet* *Select* *Reset to Ready*

Figure 3.15: Tag Access Permissions

not able to answer the *Service Request* within the defined response-time period. During calculation, the tag does not answer any other request. Therefore, the reader has to wait a certain time between sending the *Auth-Response* request and sending the *Get-Authorized-Service* request. For this application, we consider an ECDSA signing and verification time of 1 seconds and a certificate verification time of 1.2 seconds.

During the authorization process, the tag goes through the following authentication states: `NOT AUTHORIZED`, `WAIT FOR READER KEY`, `WAIT FOR TAG AUTH`, and `AUTHORIZED`. Figure 3.16 shows the state diagram for the authentication procedures.

Reader Application and Prototyping

The reader application is developed and validated with *ProtEx*. The *ProtEx* tool provides mandatory and optional reader requests as well as custom requests to develop the reader application. The reader application implemented for the MedAssist application is able to test all tag features automatically. Figure 3.17

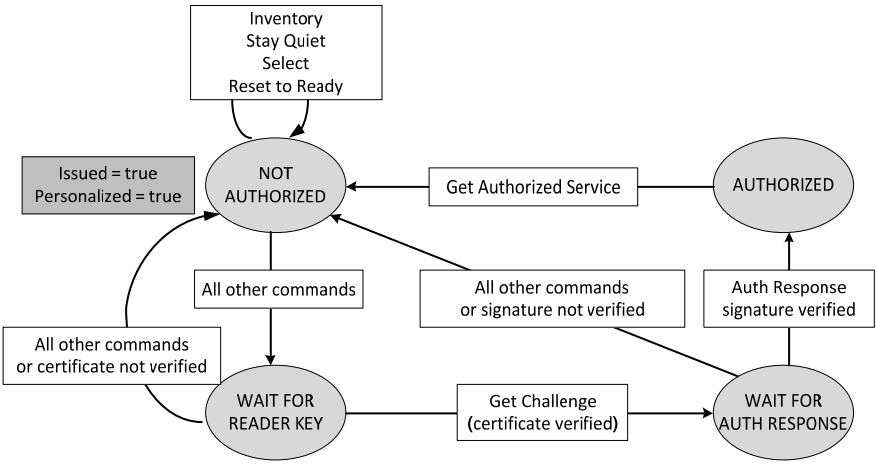


Figure 3.16: Authentication State Diagram for MedAssist Tag

shows a screenshot from *ProtEx* while testing the tag functionality. On the left side of the programm, the application flow is shown, where breakpoints are documented and the flow can be controlled by the user (application simulation to next breakpoint, simulate all, restart application aso.). The middle part of the console shows the tag status. Here, the user can observe the internal status of the tag. The left part of the program shows the log panel, which prints out the log messages scheduled by the user when writing the program code. In the figure, a successful tag testing can be seen, which ends up with an "*All Tests Passed*" message.

The next step in prototyping is the design of a tag in hardware that provides the same functionality as the tag model in software. For this purpose the IAIK *DemoTag* can be used. This device is a hardware module with an RFID interface and a programmable microcontroller. The microcontroller determines the request handling of the tag, the RFID interface receives and transmits RFID frames conforming to various RFID communication standards. The firmware of the microcontroller has been developed in a way, that the most common requests of ISO-18000-3 can be handled by the tag. It provides more or less the same functionality as the basic software tag in *ProtEx*. Therefore, the code for programming the additional functionality in the software tag can be transferred to the firmware of the physical tag. The tag firmware is written in C, so the code has to be translated from Java to C. The reader application developed for the software tags can also be used to test the tag prototypes in hardware. While writing this thesis, prototyping of the MedAssist application is in progress. Therefore, results cannot be presented at this point.

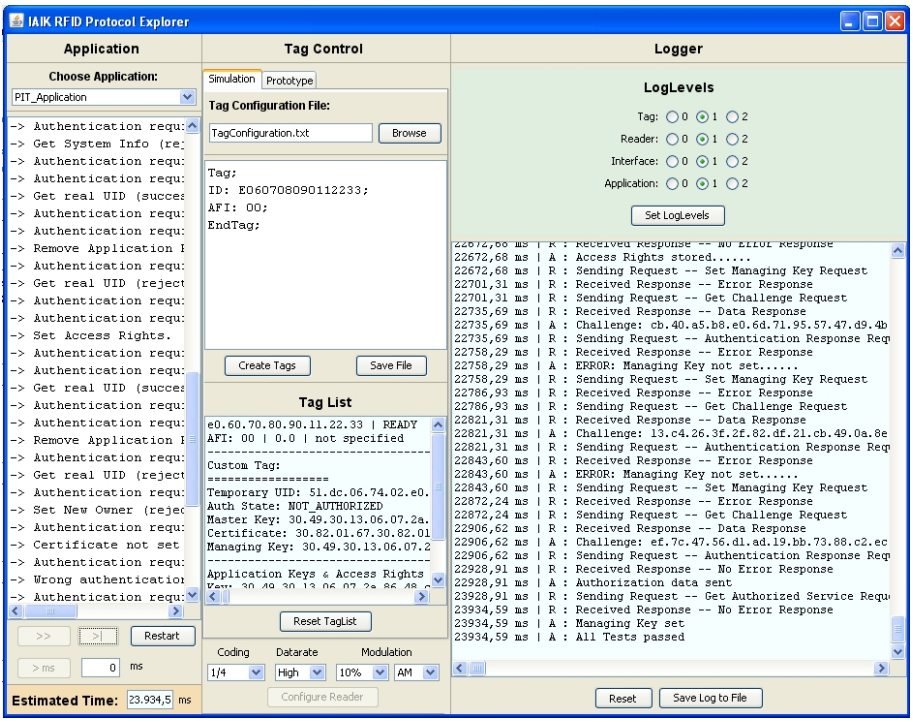


Figure 3.17: Screenshot from Testing the Functionality of the Software Tag

3.5 Conclusion

In Chapter 3 we discuss the development of three particular RFID applications including security measures. In all these applications, authentication is used to provide either (location or data) privacy, cloning and counterfeiting prevention, or access control. We show that authentication using the AES algorithm is feasible for different RFID communication standards. In Section 2.1 we have found cryptographic primitives, like AES or ECDSA, that can be used in passive RFID tags. Based on these findings, we describe sample applications including symmetric and/or asymmetric authentication. Symmetric authentication is favorable in closed-loop systems and provide a higher performance. Asymmetric algorithms can be used in open-loop systems as they provide an easier key management and better scaling capabilities.

In Section 3.4.3 we discuss the development of a secure RFID application starting from an application specification up to the design of a software model for a prototype tag. This description serves as an example for a typical application design flow. Based on the experiences from the development of the applications described in this chapter, we have defined a design flow consisting of six design phases for secure RFID applications. We describe this design flow in detail in the next chapter. During the development and evaluation of the application

protocols, we use simulation and prototyping tools to verify and adapt the functionality of the protocol and to evaluate its performance. In Section 4.2.1 and Section 4.2.2 we deal with simulation and prototyping tools, which can be used to speed up the application-design process.

4

The Design of Secure RFID Applications

Security is a requirement in many RFID applications. In the last two chapters we have discussed cryptographic primitives and protocols which have been designed for the use in RFID systems. In this chapter we go a step further and deal with the development of secure RFID applications. This task involves various phases starting from a first idea up to the production of RFID components.

During the work on the applications described in Chapter 3 we have evolved a set of best practices for the development of secure RFID applications. In Section 4.1 we classify the phases of the development process in a formal design flow. The design flow pays special attention to the integration of security mechanisms into RFID systems.

Simulation and prototyping are very important tasks during the application design flow. In these steps an evaluation and re-design of a system is relatively simple and not very costly. In order to ease and speed up these two steps, we have designed the simulation tool *PETRA*, and the simulation and prototyping tool *ProtEx*. We describe these tools in Section 4.2.1 and Section 4.2.2.

4.1 Application Design Flow

Developing RFID applications is a complex task, especially if they are supposed to include security measures. In order to ease this task, we have evolved a formal design flow. This design flow is based on an analysis of the development of several secure RFID applications (described in Chapter 3). It should serve as a kind of guideline and tasklist for application developers.

The design flow consists of various steps starting from the application specification down to the production of the components. From each step, a feedback

loop is possible to adapt former steps. In most cases, several iterations through various steps of the design flow have to be done before functional models of the application and its components can be implemented. The later in the design flow the feedback loop is used, the more costly is this step. Therefore, it is favorable to use feedback loops in the first few steps and avoid modifications in the last ones. Figure 4.1 shows the design flow. In the following, we describe the steps of the design flow in detail.

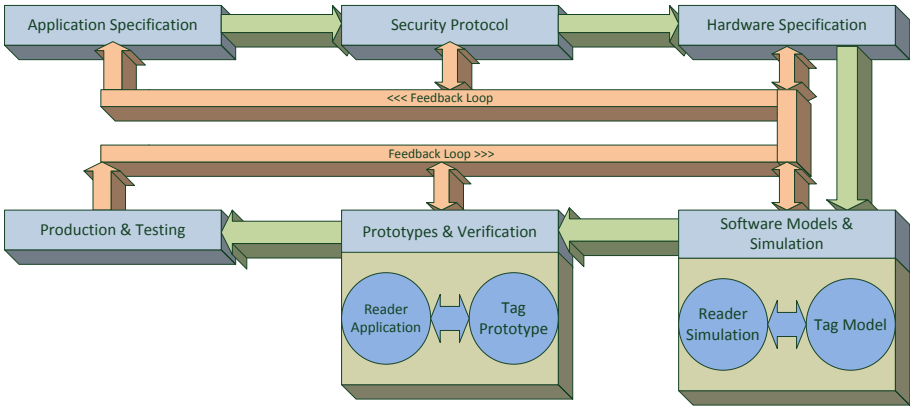


Figure 4.1: Design Flow for Secure RFID Applications

4.1.1 Application Specification

The starting point of each application development is the application specification. The result from the application-specification step is a document containing all information about the application. In the following, we describe the content of this document. The most important point is the definition of the purpose. This means that the developer has to specify which service should be provided to a particular group of persons. Furthermore, she has to identify the advantages that this group can draw from making use of the service. In the specification, all participants in the system are described with their responsibilities and capabilities.

The next task should be the definition of the interaction between the participants. In this step the communication between the participants is described in an informal way. It is defined at which point during the application flow which data is transferred from one participant to another. Also, the data processing that has to be performed by each participant to execute the application is specified. The particular implementation of the communication flow is defined in the next step, the security-protocol definition.

For RFID applications it is especially important to consider the reading range of the application. In short-range applications the tags are brought very close (a few centimeters) to the reader for communication. In long-range applications the reader can communicate with the tags even if they are a few meters

away. Another point to consider is the performance of the system: The application developer has to specify how many tags the reader is able to process in a certain amount of time. Based on the required reading range and performance of the application, the RFID communication standard (e.g., ISO-18000-3 or ISO-14443) that should be used for implementation of the application can be defined.

4.1.2 Security Protocol

The security-protocol step in the presented design flow involves the development of the general communication protocol for the participants as well as the definition of the applied security measures. The general communication protocol is based on the interaction definition in the application-specification step. The defined data flow is translated in a formal communication protocol, specifying the format and structure of the exchanged data frames.

Security measures are implemented upon the communication protocol. In order to find appropriate security measures, a threat analysis of the application has to be performed. The potential risks and attackers of the application are identified. The threats are categorized according to the damage they could cause and to their probability to happen. The application developer has to decide which threats she wants to avert and which to neglect. The outcome of the threat analysis is a specification of the parts of the communication protocol that should be secured. Based on these findings, the application developer analyzes security mechanisms that can be used as countermeasures to the defined threats.

We have already explained the relation between threats in RFID applications and security services that can be provided for RFID systems in Section 2.3.4. After the security services, e.g., proof of origin, reader authentication, or transfer of ownership, have been identified, the application developer defines the cryptographic algorithms and protocols she wants to use to provide these services. When searching for security algorithms, the developer has to consider the feasibility of the cryptographic primitive for RFID components.

As mentioned in Section 2.1.2, there are two approaches for RFID security: Using light-weight proprietary algorithms, or using standardized algorithms and protocols. We apply the second approach in our developments, as it offers a higher exploration level of the primitives and therefore stronger security. Furthermore, it is always desirable to use standardized algorithms for the sake of interoperability of different systems. We have presented standardized cryptographic primitives which provide important security services for RFID applications in Section 2.1. Furthermore, we have described how to use these primitives in standardized security protocols in Section 2.3 and Chapter 3. After definition of the cryptographic algorithms, the security protocol is integrated into the communication protocol. The outcome is a security-enhanced communication protocol, defining the interaction between the participants in detail.

After definition of the security-enhanced protocol, the proposed communication protocol has to be investigated with respect to the implementation in the used RFID communication standard. The RFID communication standard,

e.g., ISO-18000-3 or ISO-14443, has already been defined in the application-specification step. In the security-protocol step, new reader requests and tag responses are specified according to the command structure of the communication standard. Furthermore, the use of these requests and responses within the standard has to be defined at this point.

Another point to consider is the security policy on the data used in the application. Here, the developer has to decide which data has to be protected and which can be published. Based on these considerations, a specification of the data storage and access policy is developed. In this specification, the locations where data can be stored are described. Furthermore, the access rights of the applications participants to these data locations are defined. The output of the security-protocol step is an extended communication protocol formally defining interactions between the participants. The extension of the protocol includes the defined security protocols and algorithms. Furthermore, the data-access policy is defined.

4.1.3 Hardware Specification

The hardware-specification step consists of a detailed specification of the target platform. According to the required reading range and performance of the applications, an appropriate RFID communication standard has already be chosen in the application-definition step. For this communication standard, new commands have been defined in the security-protocol step. Considering the new commands, the protocol and the used cryptographic primitives, the following constraints have to be estimated: Memory usage, power, timing and size.

The application developer has to find target platforms for the reader and the tags that can be used for building the real-world application. These hardware platforms have to meet the constraints estimated before. One important point when estimating the constraints is the selection of a hardware module, implementing the required cryptographic primitive. Cryptographic operations are in many cases complex and time-consuming and are therefore very demanding for a passive RFID tag. Nevertheless, even very demanding cryptographic operations are feasible for RFID tags, as we have already discussed in Section 2.1. Another approach to provide cryptographic operations for RFID tags is the design of a new hardware module. The design of the hardware module is out of scope for the application developer. Therefore, she cooperates with a digital designer. The developer defines the requirements for the hardware module imposed by the application. The digital designer tries to meet these requirements in her development.

Another task in this step is the definition of a prototyping platform that is appropriate for testing the protocol in hardware. It should be highly probable that a protocol implemented on the prototyping platform will also work on the real-world hardware platform. The output of this step is the definition of a the destination tag and reader platform used in the real-world application, as well as the specification of the hardware used for prototyping. The definition of the

hardware platform also includes the selection of an appropriate cryptographic-hardware module.

4.1.4 Software Models & Simulation

In this step, the security-enhanced communication protocol containing new RFID commands are implemented in software. The software models of reader and tags are used to simulate their behavior during execution of the specified application. The functionality of the components as well as the protocol flow are validated and evaluated. The reader can make use of the newly defined requests that comply with the selected RFID communication standard. The tag handles these requests and returns an appropriate response defined in the application protocol to the reader.

The developer implements the tag and the reader behavior as software models and simulates the interaction between these models according to the application protocol. The software models also include the simulation of the used RFID communication standard, as the protocol is built upon this standard communication. By the use of pre-defined commands and the simulation environment provided by simulation tools, the implementation process of the software models can be sped up significantly. We describe tools providing this feature in Section 4.2.1 and Section 4.2.2.

Timing is an essential point to consider when validating the functionality of a communication protocol. Especially the time the tag needs to perform complex calculations is often not negligible. The delay that is inferred by the tag has to be taken into account during simulation. If this delay is neglected, the protocol can possibly not be implemented on the real-world hardware platform. If the tag-calculation time exceeds the time interval when the reader expects a response from the tag, alternative protocol-handling mechanisms (like interleaved protocols, see Section 2.1.1 or Section 3.1) have to be implemented. It is favorable to use protocol modifications that comply with the RFID communication standard used in the application.

Although commands and responses have already been defined in detail in the security-protocol step, implementation of the protocol can be difficult or even impossible for the target platform. Also, more efficient protocol-execution strategies can be developed during simulation. Therefore, a feedback loop to the former steps is very important at this point. Modifications done to the protocol and/or target platform can be performed less costly than in the following steps. The output of the simulation step are functional software models of the reader and the tags implementing the defined communication protocol, as well as a simulation of the interaction between these models. The simulation is used for validation and evaluation of the security-enhanced communication protocol and can lead to a possible re-design of the communication protocol.

4.1.5 Prototypes & Verification

Based on the software models and simulation done in the previous step, hardware prototypes of the components are implemented. The prototyping platform has already been specified in the hardware-specification step. The implementation of a reader prototype is easy in most of the cases. RFID readers are controlled by an application software, which in general provides the features required to implement a new application protocol. This task can even be facilitated by re-using as much as possible from the application code used in the reader software model. In very complex protocols, a modification of the reader firmware can be possible.

Prototyping is in general more costly for the tags than for the reader. The tag behavior is often implemented as a fixed state machine that cannot be modified after production. The manufacturing of a tag prototype providing new functions is expensive and time-consuming. Therefore, it is best practice to use a programmable evaluation platform for tags, like the IAIK *DemoTag*. In this case, only the firmware of the tag has to be adapted to provide new functions. The IAIK *DemoTag*, for instance, can emulate various RFID communication standards and behaves like a "real" tag in the reader field. The processing of the data on the tag can be investigated and debugged. Another advantage is that the program code from the tag model in software can be re-used up to a certain point for programming the firmware. One important point to consider when using programmable tag emulators is the matching of the target platform: The programmable tag should meet the physical constraints of the target tag platform (memory, power, size). These constraints have to be considered in the hardware-specification step, where also the target prototyping platform is defined.

If no programmable tag can be used, the prototyping of the tags has to be delayed to the production step, where new hardware chips for RFID tags are designed and produced. In this case, the software models should be exhaustively tested before reaching the production step because a re-production of new tags is expensive. The production of prototype chips is also necessary after prototyping with a programmable platform, but the probability to get a functional prototype chip in the first run is much higher in this case.

After reader application and tag prototype are implemented on the prototyping platform, their interaction is verified. A reader controlled by the application software should be able to perform the specified communication protocol with the tag prototype. We call the components, required to perform the verification, the verification environment. The output of the prototyping step is a set of functional reader and tag prototypes which are able to perform all interactions specified in the communication protocol.

4.1.6 Production & Testing

If the hardware prototypes have passed the verification, the tag circuit for the application can be produced. Therefore, a digital designer implements the spec-

ified (and tested) tag functionality in hardware, which is a very time-consuming and expensive task. The analog frontend of the tag can in general be re-used from existing solutions. The design, integration and production of the tag chip is performed following a digital design flow and using a tool chain defined by the manufacturer of the chip. This task is out of scope for the application developer. The first production lot will be small (only a few chips) to perform various tests with the new RFID tags. In many cases, the verification environment set up in the prototyping step can be re-used for testing. If the testing is successful mass production of the circuits can be started.

Readers will in general not have to be re-designed as the standard communication protocols they provide offer the possibility to extend the command set of the reader. So, only the application software controlling the reader has to be provided to the real-world readers supporting the application. This software has, in the best case, already been developed in the previous steps.

Table 4.1 gives an overview of the steps in the RFID-application design flow. It lists parameters that are considered in each design step, and the output that can be at best expected from each step. In the next section we describe prototyping tools, which can be used in the simulation and prototyping steps of the design flow to speed up these processes.

4.2 Tools for Simulation and Prototyping

In the previous section we have described the design flow for secure RFID applications. The simulation step as well as the prototyping step are crucial phases in the design process. These phases can be significantly simplified and sped up by the use of simulation and prototyping tools. In this section, we describe two simulation and prototyping tools we have developed during the research on secure RFID applications.

The first simulation tool is called *PETRA*, which has already been mentioned in Section 3.1. *PETRA* is able to simulate the behavior of tags conforming to the ISO-18000-3 RFID standard. This tool is mainly used for verification of the functionality and the evaluation the performance of RFID-application protocols. We describe *PETRA* in detail in Section 4.2.1. The code of *PETRA* has been used as the basis of another tool that is able to perform simulation as well as prototyping of RFID applications. The code developed for simulation can be re-used for implementation of prototypes in hardware. In Section 4.2.2 we describe this software, called *ProtEx*, in detail and show how to integrate the tool into the design flow.

4.2.1 *PETRA* – Simulation of RFID Protocols

PETRA is a software designed for the simulation of RFID systems conforming to the ISO-18000-3 communication standard [Int04b]. The acronym *PETRA* stands for "*Protocol Evaluation Tool for RFID Applications*". In the following

Table 4.1: Overview of the Design Flow Steps

Step	Input	Output
Application Specification	Purpose, Participants, Performance, Reading Range	Specification Document: Functionality of Participants, Interactions between Participants, Data Processing, RFID Communication Standard
Security Protocol	Interactions, Data Processing, Threats, Countermeasures, Cryptographic Primitives, RFID Communication Standard	Protocol Specification: Communication Protocol (including Security Algorithms), Data Storage and Access Policy, New Requests, New Responses
Hardware Specification	RFID Communication Standard, Cryptographic Primitives, Physical Constraints	Specification of: Destination Hardware, Cryptographic Hardware Modules, Prototyping Hardware
Software Models and Simulation	Reader Functionality, Tag Functionality, Communication Protocol, Timing, RFID Communication Standard	Reader Application, Tag Model, Simulation Environment
Prototypes and Verification	Reader Application, Tag Prototyping Platform, Physical Constraints	Reader Prototype (implementing Reader Application), Tag Prototype, Verification Environment
Production and Testing	Reader Prototype, Tag Prototype	Tag-Chip Prototypes, Reader Application Software, Test Environment, Real-World Tags and Readers

we describe the structure of *PETRA*, which is based on the structure of a real-world RFID system.

A standard RFID system consists of an RFID reader and various RFID tags entering and leaving the reader field. Reader and tags communicate using radio frequency via an air interface, where data and power is transmitted. Synchronization is also done over the radio-frequency field. The host is connected to the reader and executes the application. If necessary, the host sends reader commands and the reader issues RFID requests to the tags available in the field. The reader processes the responses of the tags and passes the result over to the host, which is able to use these data for building the application. Figure 4.2 demonstrates the structure of a standard RFID system.

The structure of a real-world RFID system can be mapped to the structure of *PETRA*, which is written in Java. Figure 4.3 demonstrates this mapping. In Java, the actors in a system are implemented as classes. In *PETRA*, the class *HostApplication* represents the host, and the *Reader* class represents the reader. The tag is mapped to the *Tag* class and the air interface converts to

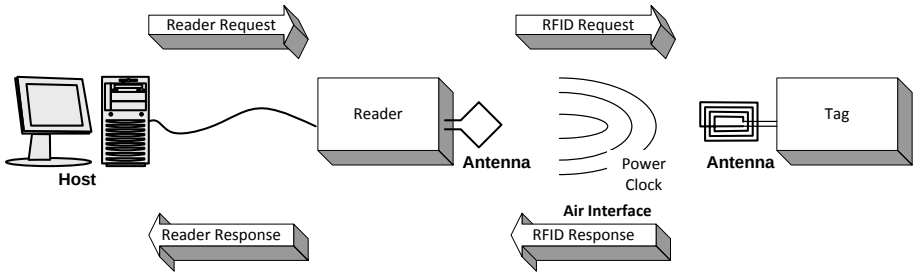


Figure 4.2: Basic RFID System

the class *Transmission Line*. The host uses *Reader Commands* to control the reader activity. The reader performs the requested action (e.g., wait, send a request, perform an inventory process). If the reader wants to communicate with the tag, it generates a valid RFID frame conforming to the corresponding RFID communication standard and sends the frame to the *Transmission Line*. The communication standard defines various requests that a reader can send to a tag. Each available request is implemented as an own Java class. In *PETRA* the following requests are implemented: *Inventory* request, *Stay-Quiet* request, and *Custom* request. The set of requests can be extended according to the user's requirements.

Tags process the received reader requests and send a response back to the *Transmission Line*. The developer can use either a *Data* response or a *No-Answer* response as reply to a reader request. The set of responses can also be extended to the user's needs. In simulation, the *Transmission Line* collects the response frames from all available tags. If more than one tag answers, a *Collision* response is generated. If no tag answers, a *No-Answer* response is returned. If only one tag issues a response, this response is relayed to the reader. The reader processes the response from the *Transmission Line*. The resulting *Reader Response* is sent to the *Host Application*, which uses the received data for building an application.

Besides the handling of the tags' responses, the *Transmission Line* class

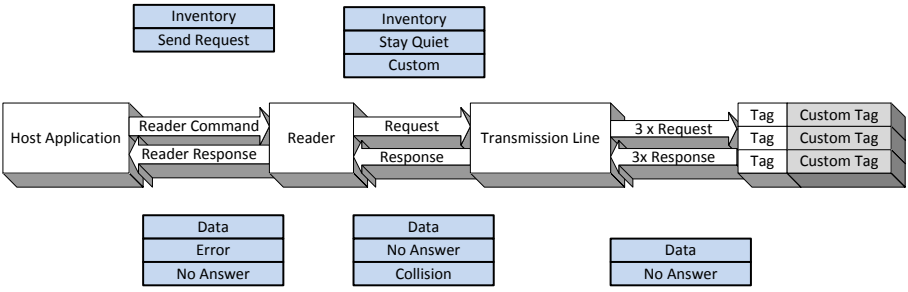


Figure 4.3: Structure of PETRA Software

processes all timing information of the system. The simulation starts at 0 ms. With each request and response, the execution time is increased. When sending a request, the *Transmission Line* calculates the time used to transmit the number of bytes contained in the request frame. Furthermore, the tag waiting time is added for a *Data* or *Collision* response. If a *No-Answer* response is issued, the time for a reader time-out is added. For each response frame, the *Transmission Line* calculates the time used to transmit the number of bytes contained. Furthermore, it derives the time the reader has to wait before sending a subsequent request. These times are added to the *Protocol Execution Time*. The *Transmission Line* always holds the current execution time of the system.

The tags also hold timing information. For each tag a *Start Time* and an *End Time* is defined. The *Start Time* represents the time when a tag enters the reader field. The *End Time* defines the time when the tag leaves the reader field. When receiving a request, the current execution time is derived from the *Transmission Line*. Each tag decides whether it is present in the reader field at the current time or not. If the tag is not available it sends a *No-Answer* response. The *Tag* class offers the basic functionality of the ISO-18000-3 standard: The tag can handle mandatory requests, namely the *Inventory* and *Stay Quiet* requests. The handling of optional and custom requests are not implemented in the *Tag* class. Nevertheless, the tag has to handle these commands as well. The user is supposed to implement additional functionality in the *Custom Tag* class. This class extends the *Tag* class.

Additional functions of the tag require an extra amount of time, that has to be considered in the simulation. Every time calculations on the tag exceed the tag waiting time, the response data is not ready at the time of sending the response. In this case, we suggest to use an interleaved protocol described in Section 2.1.1 and Section 3.1. The timing behavior of the tags can be simulated by *PETRA* by setting a *tag calculation time* for the handling of particular requests. When receiving a request from the *Transmission Line*, the tag adds the calculation time for this request to the current time and decides, if the response data can be ready at the time the response is sent. If this is not the case, the tag sends a *No-Answer* response and stores the time when response data will be ready. When the tag receives the next request, it compares the current time with the stored time. If the current time is higher than the stored time, the tag responds to the request. Otherwise, it does not answer.

The User Interface of *PETRA*

The user has to perform three tasks for implementation of software models and simulation of the application: Programming of the application, programming of the additional functionality of the tag, and configuration of the simulation parameters. In order to implement the application, the user has to modify the *Host-Application* class. She can use the defined *Reader Commands* and requests to receive data from the tags. With these data, the application can be built.

Two basic *Reader Commands* are available: One that gets the UIDs of the tags in the field (inventory procedure), and one to send a request, for instance,

Figure 4.4: Manual Configuration of PETRA Parameters

a *Stay-Quiet* or a *Custom* request. The user can define new request classes, which she can use in the application. The interface of the requests classes is designed in a way that the reader does not have to care about the structure of the request frame. The requests are translated into valid frames, corresponding to the RFID communication standard, automatically. Thus, the user does not have to care about bit- and byte-order, Start of Frame (SOF) sequence, End of Frame (EOF) sequence, or Cyclic Redundancy Checks (CRCs). She can use the requests by generating an instance of the desired *Request* class and setting the parameters (e.g., UID, data). The user can also make the application wait a certain time until issuing the next request. This time is added to the protocol execution time in the *Transmission Line* class.

The handling of additional requests (e.g., proprietary or *Custom* requests) on the tag can be implemented by modification of the *Custom-Tag* class. In order to program the tag, the user can access information about the tag state, timing, and UID, from the basic *Tag* class. By setting a tag calculation time for the handling of certain requests, the user can control the timing behavior of the tag. For generation of a valid *Data* response, the user can use dedicated methods to build a valid response frame. A *No-Answer* response is also possible. The response frame is returned to the *Transmission Line*.

When the application and the tag behavior have been implemented, the simulation of the application protocol can be performed. The specifies the simulation parameters, as tag properties, number of slots used during inventory, number of iterations, and the log-file name. Figure 4.4 shows the graphical

interface for the user to set the simulation parameters. The same parameters can also be set by using configuration files. The properties assigned to a tag are the *Start Time* and the *End Time* of the tag, and the use of an *Application Family Identifier* (AFI) byte during the inventory procedure. If the tag supports the use of the AFI byte it only responds to inventory requests containing the correct AFI code. This behavior complies with the ISO-18000-3 standard.

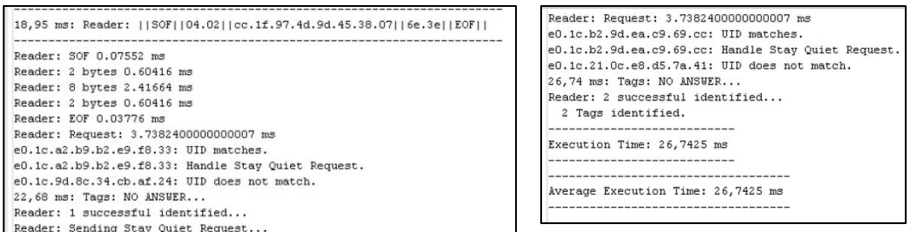
As regards the inventory procedure, the user can select to use one slot or 16 slots. The use of this parameter is specified in the ISO-18000-3 standard. In the one-slot variant, all tags answer the inventory request at the same time. In the 16-slots variant, each slot is coded with four bits. Only tags where the four least significant bits of the UID correspond to the slot number answer the inventory procedure in this slot. The number of iterations defines the number of simulation runs that are performed with the application protocol. At startup of one simulation run, all instances of the *Tag* class are initialized with a random 8-byte UID. Therefore, each run is simulated with a different tag UIDs, which results in different protocol-execution times. We explain this behavior in the next paragraph. After all runs have been passed, an average execution time over all simulation runs is calculated.

A simulation run with different tag UIDs results in different execution times even for an inventory procedure without any further protocol steps. This behavior is due to the anti-collision protocol specified in the ISO-18000-3 standard [Int04b]. In this protocol, the reader sends an inventory request and if more than one tag answer, a collision is detected. The reader can determine the bits up to the point where the collision occurs. It generates a mask with these bits, which are correctly received and extends it with one bit (either 0 or 1). Now, the reader sends a new inventory request including the mask. Only tags where the UID is equal to the mask answer this request. With different UIDs the collision occurs at different points in the protocol. Therefore the protocol execution time differs as well.

The output the user gets from the simulations is a log file. Depending on the log level, different information is logged. The log-file name and the log level are also set during configuration of the simulation parameters. In programming, pre-defined logger statements can be used to control the logged output. Figure 4.5 shows two parts of a log file. The reader and tag activities are logged in detail. The current protocol-execution time is given in each line to check the timing behavior of the application. At the end of each simulation run, the required execution time is stored. The average execution time of all simulation runs is calculated and logged at the end of the file when all runs are finished.

Supporting the Design Flow with *PETRA*

The simulation tool *PETRA* is mainly used to support step four of the design flow (*Software Models & Simulation*). The application developer can make use of pre-defined reader commands to program the host application. Basic tag functionality complying the ISO-18000-3 standard ([Int04b]) already exists



```
------------------------------------------------------------------------
18,95 ms: Reader: ||SOF||04.02||cc.1f.97.4d.9d.45.38.07||6e.3e||EOF||
------------------------------------------------------------------------
Reader: SOF 0.07552 ms
Reader: 2 bytes 0.60416 ms
Reader: 8 bytes 2.41664 ms
Reader: 2 bytes 0.60416 ms
Reader: EOF 0.03776 ms
Reader: Request: 3.7382400000000007 ms
e0.1c.a2.b9.b2.e9.f8.33: UID matches.
e0.1c.a2.b9.b2.e9.f8.33: Handle Stay Quiet Request.
e0.1c.9d.8c.34.cb.af.24: UID does not match.
22,68 ms: Tags: NO ANSWER...
Reader: 1 successful identified...
Reader: Sending Stay Quiet Request...
Reader: Waiting 0.3209 ms...REQUEST_TO_RESPONSE
------------------------------------------------------------------------
23,00 ms: Reader: ||SOF||04.02||24.f5.d3.2c.31.b9.38.07||ea.64||EOF||
------------------------------------------------------------------------
Reader: SOF 0.07552 ms
Reader: 2 bytes 0.60416 ms
Reader: 8 bytes 2.41664 ms
Reader: 2 bytes 0.60416 ms
Reader: EOF 0.03776 ms
Reader: Request: 3.7382400000000007 ms
```

```
Reader: Request: 3.7382400000000007 ms
e0.1c.b2.9d.ea.c9.69.cc: UID matches.
e0.1c.b2.9d.ea.c9.69.cc: Handle Stay Quiet Request.
e0.1c.21.0c.e8.d5.7a.41: UID does not match.
26,74 ms: Tags: NO ANSWER...
Reader: 2 successful identified...
  2 Tags identified.
---------------------------
Execution Time: 26,7425 ms
---------------------------
-----------------------------------
Average Execution Time: 26,7425 ms
-----------------------------------
```

Figure 4.5: Log-File Examples for *PETRA*

and can be used without further modifications. Additional tag functionality is supposed to be programmed in a separate Java class (*Custom Tag*). In this way, all modifications and extensions to the basic tag functionality are performed in one place, which helps to reduce complexity of the programming task.

When software models of the tag and the reader have been implemented, tag functionality and reader application are validated by simulation. *PETRA* writes a log file where the application developer can examine the reader and tag activities in detail. The logging functionality can easily be extended to the requirements of the developer. In this way, *PETRA* supports the developer in her task of validating the application.

PETRA is also used to evaluate the performance of the application protocols. The developer can set up different scenarios with different numbers of tags, which move at different times through the reader field. Also different inventory-slot numbers and AFI settings can be used to investigate on the performance. The simulation tool calculates the average protocol-execution time for each setting. In this way, different settings can be compared in regard to performance. The evaluation feature of *PETRA* is limited to the application level, which means that delays by physical effects and transmission errors are not taken into account.

The use of *PETRA* helps to speed up the prototyping process due to reusability of the code of the software models. The implementation of the host application can be used in step five of the design flow (*Protoypes & Verification*) for building the reader prototype. The implementation of the additional tag functionality can be re-used to configure the tag prototype. The degree of reusability strongly depends on the hardware platform that is used for prototyping. In the next section we describe the simulation and prototyping tool *ProtEx*, which has been developed to maximize the reusability of the application and the tag code in the hardware prototypes. When taking advantage of the similarities of both platforms, the prototyping process can be sped up significantly.

4.2.2 *ProtEx* – Joining Simulation and Prototyping

ProtEx is a simulation and prototyping tool that uses synergies between software models and hardware prototypes to speed up the RFID-application design process. The simulation part of *ProtEx* is based on *PETRA*, the simulation tool described in Section 4.2.1. *ProtEx* extends the simulation part of *PETRA* and adds a prototyping part where physical hardware is involved. Therefore, we named it *PROTotyping EXpress* (*ProtEx*). The software can control an RFID reader and a programmable tag emulator via the serial interface. *ProtEx* is designed in a way that applications programmed for simulation of the software models can be used without modifications for controlling the physical reader. The code for the software model of the tag can be to a high degree re-used in the programmable tag. As the programming languages of both platforms differ, certain modifications to the code have to be done anyway. Nevertheless, the structure and methods used in the software model can also be embedded in the tag hardware platform.

The Idea of *ProtEx*

The idea of developing a tool that is able to perform simulations and prototyping as well is based on various experiences with other simulation tools for RFID applications. Besides *PETRA* we have experienced other types of software simulation tools during our research. For the development of the application described in Section 3.3, we have designed a simulator of an NFC application. The simulation was very helpful for development and implementation of the application, but as it was programmed for this particular purpose it was not reusable for other applications.

Furthermore, we used the simulation tool RFIDSim, described in [FP08] and [FWS08]. This tool is able to simulate, besides the functionality of the components, also physical effects of radio transmission like pathloss, fading, and backscatter. RFIDSim works with the ISO-18000-6C standard, which is also called EPCglobal Gen2 standard [Int04c]. We implemented an authentication protocol on this simulator and evaluated the performance of this protocol. During the work with RFIDSim we experienced that simulation of physical effects in RFID systems adds very high complexity to the simulator and is not necessary for prototyping at application level. Furthermore, RFIDSim has not been designed for building new applications. Therefore, implementation of new protocols is not intuitive and user-friendly. The user has to examine and understand the structure of the simulation tool to add new functionality to the simulator.

In *ProtEx* we want to circumvent the disadvantages we previously described. We have derived a set of constraints and features we want to meet. Thus, we have designed our simulation and prototyping tool *ProtEx* under the the following directives:

1. The software framework is supposed to be reusable for all kind of RFID applications.

2. Basic functionality of the chosen communication standard is to be provided to the user.

3. The usage of the framework is to be as intuitive as possible. This means that, the user does not have to care about details of the communication standard or the framework structure.

4. A high degree of reusability from the software models to the hardware prototypes is to be provided.

The Structure of *ProtEx*

As *ProtEx* is based on the simulation tool *PETRA*, the structure of the program is partly similar. Figure 4.6 shows an overview of the structure of *ProtEx*. Additional to the software modules, hardware components are used that are connected to the serial interfaces and can be controlled by *ProtEx*. We use the IAIK HF RFID Reader as reader device and the IAIK *DemoTag* as tag prototype for *ProtEx*. Before starting the simulation or the prototyping process, the software models configuring the tag behavior or the firmware of the programmable tag prototype has to be implemented.

When the tag behavior in the application has been programmed, the reader application can be built. The host application uses, like in *PETRA*, *Reader Commands* to control the reader activity. The application can tell the reader to get the UIDs of the tags in the field (*Inventory*), to send a defined request to a tag, or to perform some reader configurations (e.g., reset, setting a data rate or modulation). After the host application has been implemented, the user can select wether she wants a simulation with the software modules or a prototyping test with the hardware modules.

If simulation has been selected, the commands are handled by the software model of the *Reader*. Every time communication with the tag is required, the reader sends the appropriate request to the *Air Interface*, which corresponds to the *Transmission-Line* class in *PETRA*. The *Air Interface* sends the request to all available tag instances, which are represented by the software model of the tag in simulation. Each tag handles the received request and returns a response to the *Air Interface*. The *Air Interface* collects all tag responses and preprocesses them before sending a response to the reader. If all tag instances answer with a *No-Answer* response, a *No-Answer* response will be returned. A single *Data* response or an *Error* response is simply relayed to the reader. If more than one tag response containing data has been received, a *Collision* response is returned.

If prototyping has been selected, the *Reader Commands* are translated into control frames for the physical reader. These frames are sent via the serial interface to the reader. If communication with the tag is required, the reader and the tag prototype execute the RFID communication protocol for the corresponding reader request. The response from the tag prototype is sent to the reader, which translates the response in a reader response frame. The frame is sent back to *ProtEx* via the serial interface. As the frames from the physical

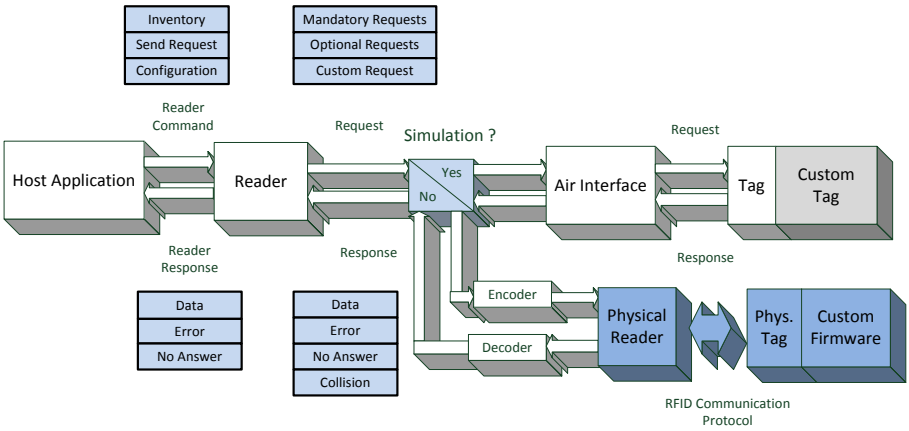


Figure 4.6: Structure of *ProtEx*

reader are different from the *ProtEx* reader responses, the frames are decoded and translated into response-frame format defined in *ProtEx*.

Independent from where the response received from the *Air Interface* originates from (software model or prototype) the *Reader* class translates the response into a *Reader Response* format, which can be further processed by the *Host Application*. This means that the user does not have to care about the platform she uses. The developer can control both simulation and prototyping in the same way by using the same commands and requests. This is the basis for the reusability of the host application in simulation as well as for prototyping.

When using the software models for simulation, timing information can be derived. Like in *PETRA*, the system has a current *Protocol Execution Time* that is updated with every request or response sent over the *Air Interface*. Furthermore, the mobility of tags can be emulated by using the tag timing parameters. The software model of the tag holds a *Start Time* and an *End Time*, which represent the times when a tag enters and leaves the reader field. Each tag receiving a request determines whether it is present in the field at the time a request is received before handling the request or not. If the tag is out of the field, a *No-Answer* response is issued.

Another timing issue, also already handled in Section 4.2.1, is the tag calculation time. If a complex process on the tag exceeds the response time until the reader detects a time out, this must also be handled in simulation. So the user can define calculation times for certain processes on the tag. If the tag performs such a process, the tag is set busy for this calculation-time period and does not answer to any further request. The tag calculates the time when the process will finish the calculation. This is done by simply adding the current protocol execution time with the calculation time. When receiving a subsequent request, the tag determines whether the tag is busy or if it has already finished the calculation. If the tag has finished the process, it responds to the request. If the tag is still busy, a *No-Answer* response is issued.

Configuring *ProtEx* for Different RFID Communication Standards

The general structure of *ProtEx* allows the implementation of frameworks for different RFID communication standards. In this section we show which steps are required to get a framework for any particular standard. In the first version of *ProtEx* we have implemented the ISO-18000-3 standard [Int04b]. Therefore, we refer to this standard when giving implementation examples in this section. For a proof of concept, we also have implemented an ISO-14443 version of *ProtEx*.

Table 4.2: Implementation Tasks for *ProtEx* for a particular RFID standard

Part	Component	Task
General	Requests	Implementation of required request classes
	Responses	Implementation of required response classes
	ReaderCommands	Definition of new reader commands (e.g., configuration)
Simulation	Reader	Implementation of basic reader commands (inventory, send request)
		Implementation of new reader commands (configuration)
		Preprocessing of the responses for the application (collision handling)
	Tag	Implementation of basic requests
		Handling of custom commands in *Custom Tag*
	Timing	Definition of timing constants
		Implementation of timing calculation for transactions in *Air Interface*
Prototyping	Reader	Implementation of missing basic functionality in firmware (rarely)
	Tag	Implementation of missing basic functionality in firmware (rarely)
	Encode/Decode	Implementation of the mapping between requests and control strings for physical readers
		Implementation of the mapping between received response strings and defined responses

In Table 4.2 we give an overview of the tasks necessary when configuring *ProtEx* for a new RFID communication standard. The first task is the definition and implementation of basic requests and responses the user should be provided with. These commands are the link between simulation and prototyping, as they are used equally for both platforms. The structure and definition of requests and responses can be found in the corresponding RFID standard, e.g., [Int04b], [Int00], [Int04a], or [Int04c].

In the ISO-18000-3 variant of *ProtEx*, we have extended the set of available

requests in *PETRA* by several optional requests (read and write operations on the tag's memory, *Select*, *Reset to Ready*, *Get System Information*). In that way, we have provided the most common functions for this standard. Nevertheless, the set of requests and responses can be even more extended by the user. The framing of the commands should be abstracted from the user. This means that the user only has to generate an instance of the required request or response class with the necessary parameters (e.g., UID, header, or data). The building of the correct frame structure for the corresponding communication standard is done automatically by the software.

For the simulation part of *ProtEx*, the tag functionality and the reader behavior have to be defined. The *Reader Commands*, which are available to control the reader via the *Host Application*, are an inventory command, and the command to send a request and return the response. At least these two *Reader Commands* have to be implemented according to the selected communication standard. The set of *Reader Commands* can be extended, for instance, if some configuration parameters can be set in the reader device. For the ISO-18000-3 standard, there exist various possibilities of modulation, data rates, and coding of the data sent or received by the reader. These parameters have an influence on the timing behavior of the reader and should be considered during simulation. Therefore, we have implemented the configuration of the reader parameters as additional *Reader Commands*. The developer can use these commands to configure the software model of the reader as well as the physical reader.

The next step is the modification of the *Reader* class. The processing of the responses received from the *Air Interface* has to be defined. For different communication standards, different pre-processing is possible. Especially, the handling of a collision has to be considered. The *Air-Interface* class returns either a *No Answer*, a *Collision* or another response defined by the user. The translation of the responses into a *Reader Response* that can be handled by the *Host Application* has to be defined.

The behavior of the tag is specified in the selected communication standard. The tag processes RFID requests and returns the corresponding RFID responses. Handling of basic RFID requests has to be implemented. Custom commands can also be provided, but the implementation of their processing is left to the application developer. For this purpose, the *ProtEx* software defines a method *handleCustomCommand(Request)* that is a mandatory method to be implemented in the *Custom Tag* class.

In order to calculate the protocol-execution time during simulation, timing parameters must be defined. In general, the programmer defines constants like the tag waiting time, the reader waiting time, and the time used to transmit one byte of data. The values of these constants can be derived from the corresponding RFID communication standard and are used during calculation of the execution time. The calculation of the current time is performed in the *Air Interface* class. For each transaction in the *Air Interface* (sending of request, receiving of response) the timing calculation has to be defined.

For the prototyping part, a physical reader and a programmable tag support-

ing the selected RFID communication protocol are required. These components are supposed to be already available before the integration into *ProtEx*. For the ISO-18000-3 variant of, we used the IAIK HF RFID Reader device and the IAIK *DemoTag*. If a particular request defined in the basic functionality of *ProtEx* is not supported by the reader or the tag, this request has to be implemented in the firmware of the appropriate device. When using standard requests and existing reader and tag devices, this case will occur very rarely.

The last task for the configuration of *ProtEx* to a particular RFID communication standard, is the definition of the link between the simulation and the prototyping part: The programmer has to define the translation between requests and the command strings sent over the serial interface to the reader (encoding). Furthermore, the mapping between the response strings received from the reader over the serial interface to defined responses (decoding) has to be specified. For this purpose, the programmer has to consult the reader's manual to find the appropriate control and response strings. Afterwards, she maps the basic requests and responses, defined in *ProtEx*, to these strings. The mapping is executed in the *Encode* and *Decode* methods of the reader.

When *ProtEx* has been configured for a particular communication standard, the application developer can implement all kinds of RFID applications using this standard. The implementation of the application is abstracted from the communication standard as the basic functionality is already provided by the tool. These features correspond to the first two points in the directive list we defined for the development of *ProtEx*(see Section 4.2.2). The task of the application developer is the programming of the additional functions of the tag and of the application by using pre-defined commands, requests and responses.

Developing RFID Applications with *ProtEx*

*ProtEx*is used for the development of new RFID applications. In the following, we describe the steps that are required to get a working simulation and prototyping environment for a new RFID application. By re-using parts from the simulation environment in the prototyping phase, the application-development process can be simplified and accelerated. In this section, we explain some aspects of the handling of the *ProtEx* software in order to show that the tool can be handled very intuitively, which was one of the design goals, namely point three in the directive list in Section 4.2.2.

Testing of Basic Functionality

For the sake of convenience, application simulation in *ProtEx* can be controlled by a graphical user interface (GUI). Before starting to implement the programming parts, the application developer can test the functionality of the selected RFID communication standard by using the testing mode of *ProtEx*. The testing mode consists of a GUI which enables the user to send requests manually, either to the simulation platform or the prototyping platform. Figure 4.7 shows the GUI for testing mode. The command panel on the left side of the GUI offers

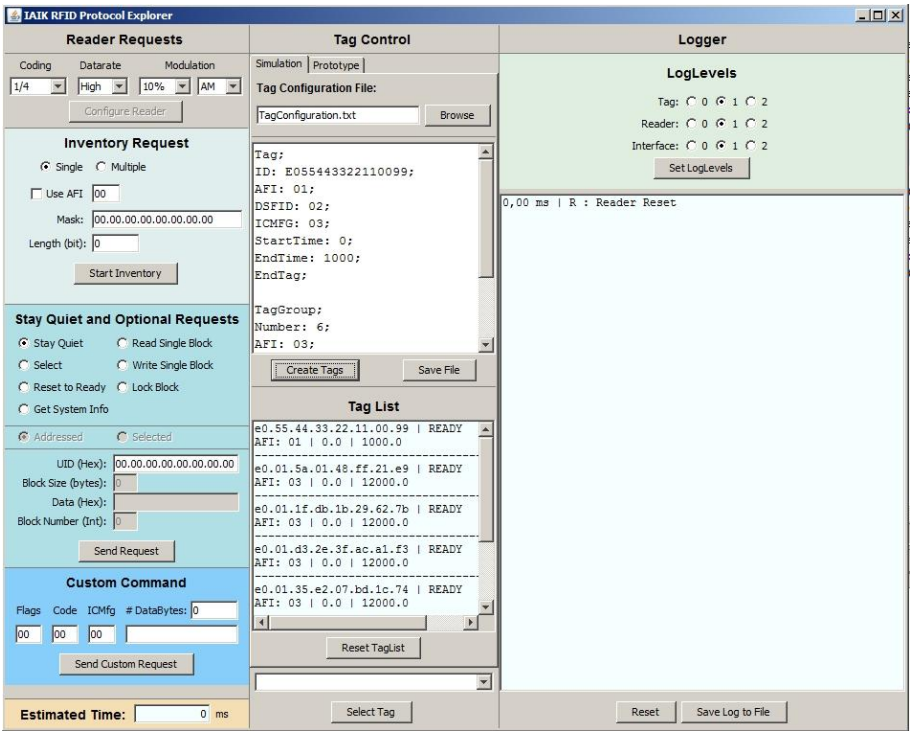


Figure 4.7: GUI for Testing Mode of *ProtEx*

the user the possibility to issue an inventory command with different parameters (slots, AFI, mask), to send a defined or custom request, or to modify the reader configuration. When selecting a request, the required input parameters for the request are enabled. The parameters that are not required are disabled. The functions accessible in this panel show the basic functionality implemented in *ProtEx*. These functions have already been pre-defined for convenience of the user.

The tag-control panel is located in the middle of the GUI. In this panel, the user can select either the simulation or the prototyping platform by clicking on the corresponding tab. In Figure 4.7, the simulation platform of *ProtEx* is selected. The user defines tags in the text area by using the syntax shown in the figure. Alternatively, she can load the tag configuration from a file. The content of the text area can be stored in a file for future use. The user can define single tags with a defined UID and/or a group of tags (with the same properties) with random UIDs. By pressing the button "Create Tags", the tag instances with the defined configurations are initialized for simulation. The status of all created tags is displayed at the bottom of the tag-control panel.

If the prototyping platform is chosen, the tag-control panel consists of a GUI for controlling the IAIK *DemoTag* and two areas that show the current status

of the tag. Figure 4.8 shows the tag-control panel for the IAIK *DemoTag*. The user can, for example, set a new UID to the tag, change the communication standard for the tag, set the AFI, or refresh the status information for the IAIK *DemoTag*. The first status area displays the general settings and the state of the tag. The second area shows the messages received and sent by the tag.

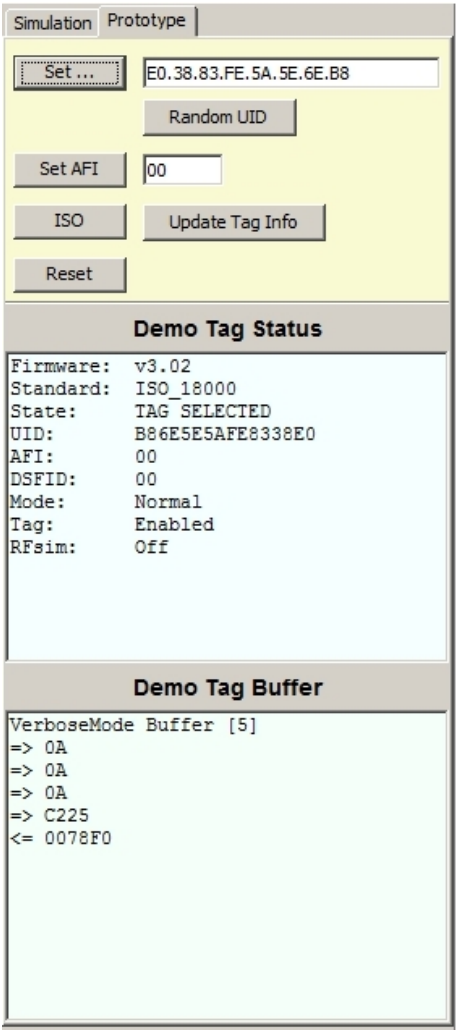


Figure 4.8: Tag Control Panel for IAIK *DemoTag*

After configuration of the software tags or the connection to the physical reader and the IAIK *DemoTag*, the command panel can be used to issue reader commands and send requests to the tags. The communication between reader and tags is logged in the logging panel on the right side. Here, the user can examine the communication flow, the request handling, and the communication

frames as well as the timing. The amount of details in the logging messages can be controlled by configuring the log levels at the top of the logging panel. At the bottom of the command panel, the estimated protocol-execution time is displayed. At the bottom of the tag-control panel, the user can find a drop-down list of all identified UIDs. By pressing the button "Select Tag" under the list, the selected UID is written in the UID field of the command panel, in order to ease the selection of a particular tag UID.

Implementation and Testing of Custom Commands

If the application developer is already familiar with the RFID communication standard, she can start to implement the additional functionality for the new RFID application. For this purpose, new requests and responses are defined. Each new command is implemented in an own Java class. The main part of this step is the programming of the additional functionality of the tag. The handling of new requests is implemented in the *Custom Tag* class. After programming of the new functionality, the testing mode can be used once again to manually check the behavior of the custom configured tag.

It is advisable to use the software model of the tag for validation of the new functionality at this stage. The programming of the physical tag is done after the complete application involving the software tags has been validated. Modifications are likely to be done to the tag functionality after the application-testing phase. The software model can easily be modified, but modifications can be even more effort when they have to be implemented on the hardware platform.

Implementation and Simulation of Host Application.

Programming a new application in *ProtEx*, the application developer creates a new application class, which extends the *RFID Application* class. The user can send *Reader Commands* (inventory, send request, configure reader) to the reader. The reader can send all pre-defined and additional requests to the tag. Sending of a request results in a tag response, relayed from the reader to the host application, which contains information that are used to build the application. Figure 4.9 shows a very simple example of a host application, where an inventory request is issued and a *Stay Quiet* request is sent to all identified tags.

In the application code, some additional commands for simplifying the debugging of the application code can be used. The user can apply logging commands, which output logging messages in the logging panel. The user can determine at which log level the message is written on the panel. Furthermore, the developer can define breakpoints in the code. These breakpoints are used to control the application flow during simulation. *ProtEx* is able to pause the execution of the application at these breakpoints. When defining a breakpoint the user assigns a text to the breakpoint which is displayed in the *Application Panel* when the breakpoint is reached. Figure 4.10 shows the application

```
package at.iaik.hf15693protex.custom.application;

import java.util.ArrayList;

public class TestApplication extends RFIDApplication {

    public TestApplication(){
        super();
    }

    protected void application() throws InterruptedException {
        Logger.writeln("Test Application", LogType.APP, 1);
        if (reader_ != null) reader_.sendInventoryRound(new byte[]{0x00}, (byte) 0x00,
                                                        false, false, (byte) 0x00);
        ArrayList<byte[]> tagList = reader_.getInventoryResponse();
        waitWhileSuspended("Inventory done.");
        HF15693StayQuietRequest stayQuiet;
        for (byte[] tag: tagList) {
            stayQuiet = new HF15693StayQuietRequest(tag);
            reader_.sendRequest(stayQuiet);
            waitWhileSuspended("StayQuiet sent.");
        }
    }
```

Figure 4.9: Sample Application Code for *ProtEx*

mode of *ProtEx*. On the right side, the command panel is substituted by the application panel. The other panels stay the same as for testing mode.

Using the application panel, the customized tags can be tested with the host application. In the upper part of the application panel, the available applications are displayed. All of them are Java classes that extend the *RFID Application* class and are placed in the appropriate directory. The buttons at the bottom of the panel (“`>>`”, “`>|`”, “`->`”, “`>ms`”, and “`Restart`”) control the application execution. The application can be executed until the end of the code is reached. Also an execution to the next breakpoint or an execution over a certain amount of time is possible. The protocol execution time is logged at the bottom of the application panel.

By examining the application status and the logging panel, the developer can verify the functionality of the customized tags and validate the application protocol. During this phase, modifications in the behavior of the tags or the application flow can easily be performed by adapting the source code to the requirements of the RFID application. When the developer has sufficiently tested the application and is satisfied with the behavior of the software models, she can go to the next step: The implementation of the prototypes.

Implementation and Verification of Prototypes.

When the software model of the tag has been validated, the same functionality should be implemented on the hardware platform. The application developer has to transfer the code from the *Custom Tag* to the tag prototype. In our case, the tag prototype is the IAIK *DemoTag*, which is a programmable hardware platform. The firmware of the tag prototype is written in C. Thus, some

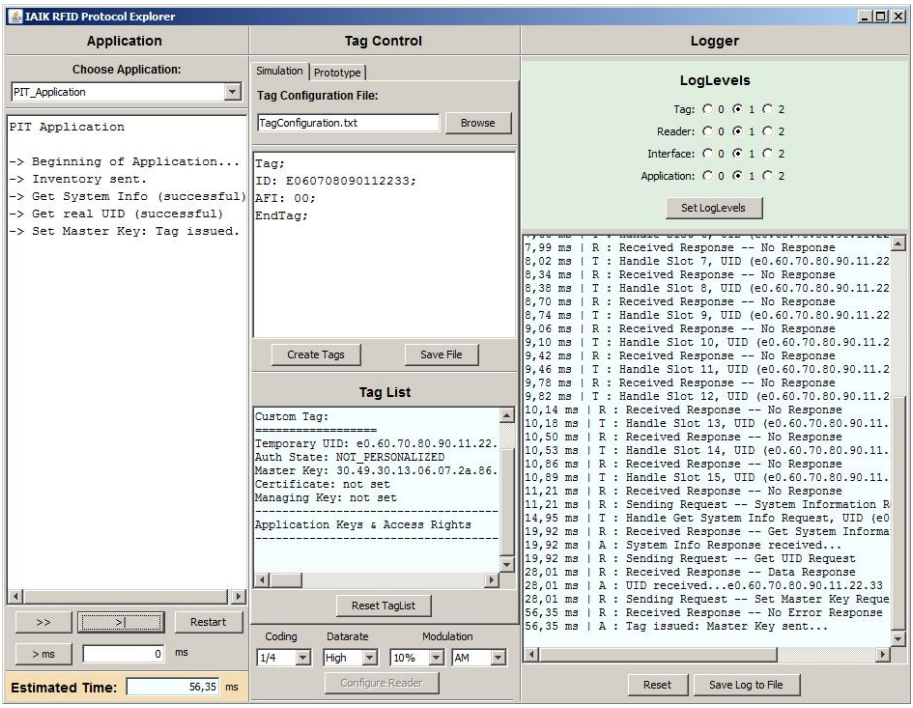


Figure 4.10: Application Mode for *ProtEx*

adaptations have to be done to transfer the Java code of the *Custom Tag* to the firmware of the IAIK *DemoTag*. Nevertheless, the firmware of the IAIK *DemoTag* is written in a way, that custom commands can be implemented separately from the basic functionality. This is also true for the tag software model in *ProtEx*. All necessary modifications can be found in the *Custom Tag* class. Therefore, the implementation of the additional functionality in the software module can be used as the basis for coding the new commands in the tag prototype.

The customized tag prototype can be tested either manually in the testing mode or can be automatically verified with the host application. The host-application code does not have to be changed, as the same reader commands and requests are used to control the software model of the reader and the hardware reader. This means that, the application code is completely reusable for the prototyping phase. Verification of the prototype functionality with the application is performed in the application mode of *ProtEx*Ḧere, the prototyping tab is selected and the existing application code (already used for simulation) is executed. Controlling the application flow also works the same way as for simulation. The output of the logging panel shows the communication flow between the hardware devices. In the tag control panel, the status and the message buffer of the IAIK *DemoTag* can be examined.

Supporting the Design Flow with *ProtEx*

In this section we have described the simulation and prototyping tool *ProtEx* for developing RFID applications. *ProtEx* can be used in the *Software Models & Simulation* step and in the *Prototypes & Verification* step of the design flow. During simulation, the tool is used to adapt the application protocol and tag behavior very quickly to the application requirements. By providing the basic functionality of the selected communication standard to the application developer, the implementation of the tag software model and the reader application is simplified and sped up. Furthermore, a performance evaluation of the protocol can be done before even hardware is involved.

In the prototyping phase the code from the tag software model can be used as basis for programming the tag prototype. The application code can be completely re-used for verification of the prototypes. By using large parts of the software model and the application code for the prototyping and verification phase we provide a maximum of reusability, which corresponds to point four in the directive list in Section 4.2.2. After verification of the functionality of the tag prototype, the prototyping phase is finished and further steps like production of a chip prototype can be started.

4.3 Conclusion

In Chapter 4 we deal with the design process of secure RFID applications. We describe a design flow including six design steps starting from the application specification up to the production of tags. The second step in this design flow represents the design of the security protocol. During this step, cryptographic algorithms and hardware modules, described in Chapter 2, are used for providing security services to RFID applications. We have described some examples of the development of security protocols for RFID systems in Chapter 3.

Simulation and prototyping are important steps in the design flow. During these steps, modifications to the application protocol can be done with low effort. Therefore, the application developer is to be supported especially during simulation and prototyping. We have designed the simulation tool *PETRA* and the simulation and prototyping tool *ProtEx* for this purpose. We describe these tools in Section 4.2.1 and Section 4.2.2. The developer is supported by predefined basic functionality for the RFID communication standard that can be used for application programming and for implementation of the tag software model. Using the software *ProtEx* a high degree of reusability of the reader application and the tag code is provided. In this way, a fast prototyping process can be triggered.

This chapter finishes the first part of the thesis, where we discuss three important factors of designing secure RFID applications: Cryptographic hardware, security protocols, and the integration of the protocols into secure applications. In many secure applications communication with other remote devices on the Internet can add an extra value. With this capability also new RFID applica-

tions will be enabled. Therefore we dedicate the second part of the thesis to the connection of RFID tags to the *Internet of Things*. In the following chapter we describe an approach, how passive RFID tags can communicate using the Internet Protocol version 6 (IPv6). In Chapter 6 we develop a security layer for the online communication.

Part II

Passive RFID Tags on the *Internet of Things*

5

Connecting RFID Tags to the Internet

In Part I, we have discussed the design process for secure RFID applications. In this part, we describe how to securely connect RFID tags to the *Internet of Things*. As RFID tags have evolved from a mere bar-code replacement to computing devices, new applications including complex functionality are enabled. Taking advantage of tag capabilities using a remote connection over the Internet will be a real advance. Remote communication with other devices and exchange of data over the Internet can be used for building new applications.

In this chapter we present an approach to connect passive RFID tags to the Internet without adding too much workload on the tags. As the common "language" of the Internet is the *Internet Protocol* (IP), we base the proposed communication concept on this protocol. As there will be a huge amount of tags and other mobile devices on the *Internet of Things*, we use IP version 6 (IPv6), which has a large address space. In particular, we use mobility concepts defined for IPv6 (Mobile IPv6, respectively) where tags can be considered as mobile nodes on the Internet. In this way we enable two-way communication between any node on the Internet and passive RFID tags. In the following we describe the basic concept on the *Internet of Things*. We present basics of Mobile IPv6 (MIPv6), and show how passive RFID tags can "talk" IPv6 with other participants on the *Internet of Things*.

The chapter contains results of joint work together with Manfred Aigner and Stefan Kraxberger. Parts of the outcomes were published at the International Conference for Internet Technology and Secured Transactions 2010 [DAK10], at the RFIDsec Asia Workshop 2011 [DGAK11], and in the Journal of Security and Communication Networks, Special Issue on Protecting the Internet of Things [DK11].

5.1 RFID and the *Internet of Things*

Nowadays, mobile devices containing microprocessors pervade everyday life of most people. Objects which we do not perceive as computing devices act like computers. In most cases, these objects are everyday objects constantly around us. They can process, receive and send data and can even connect themselves to other objects. Many mobile devices are able to measure and "sense" their environment. In an important essay, Mark Weiser has introduced the term *ubiquitous computing* already in 1988 [Wei91]. He described ubiquitous computing as follows: "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. [...] a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background. [...] Indeed, the opposition between the notion of virtual reality and ubiquitous, invisible computing is so strong that some of us use the term "embodied virtuality" to refer to the process of drawing computers out of their electronic shells.*"

In 2000, Sarma et al. presented their vision of an *Internet of Things* based on RFID-tagged items [SBA00]. Their system allows tracking and tracing of items by individual identification of items on basis of low-cost RFID tags. Now, years later, this vision has already become reality. Mobile communication, Internet, as well as RFID technologies are omnipresent. Regardless of the protocols they use, objects are connecting and communicating to other devices. They exchange data and learn more about their environment. These features enable the objects to act like they were "smart". Even RFID tags, which represent one of the most restricted devices on the *Internet of Things*, already offer advanced functionality and they are evolving.

The concept on the EPCglobal network described in [EPC07] is the first step towards integration of RFID technology into the conventional Internet. Data collected from RFID tags are sent to servers on the Internet, where they can be retrieved by other nodes. The EPCglobal network defines a set of services that are similar to services on a traditional IP network (e.g., naming service, discovery service, information service, and security service). The RFID tags used in this system are EPC (Electronic Product Code) tags specified in [EPC03] or [Int04c]. Within the EPCglobal network, RFID tags play a passive role. They do not interact with other nodes or items on the Internet.

Our aim is to extend the communication capabilities of passive RFID tags. In the proposed system, RFID tags are able to interactively communicate with other participants on the network. In this way, RFID tags are integrated as active parts into the *Internet of Things*. The possibility to interactively communicate with tags as well as the increased functionality of tags will lead to applications, where, for instance,

- an authorized party can remotely access a tag over the Internet in order to, for instance, poll information or write new data on the tag,
- an RFID tag can request information from a remote server,

- or an RFID tag can communicate with other RFID tags.

Examples for such applications are guarantee and maintenance management, which can be done over the Internet. Another possible scenario is remote revocation of RFID tags, for example, during passport control at the border. Online access management is also a application scenario: The RFID tag requests information about the reader from a remote server. From the server response, the tag can decide which information to reveal. It is difficult to predict the whole variety of future applications. However, the capability to interactively communicate and exchange information with other parties on a network is a tremendous change in paradigms for the use of passive RFID tags.

To make a long story short, the *Internet of Things* can be considered as a huge extension of the traditional Internet. All computing devices that are able to communicate are connected to this network. One main concern of this vision is that the devices are very heterogeneous and communicate in many different "languages" (communication protocols, respectively). So the following questions arise:

- What should be the common communication protocol of the *Internet of Things*?
- How can devices learn to communicate conforming to this protocol?
- Which kind of devices can be integrated into the *Internet of Things*?

As already mentioned, the Internet Protocol is the common "language" of the conventional Internet. Hence, it is highly probable that it also will be established as the standard communication protocol of the *Internet of Things*. There are two possibilities how devices can be adapted in order to communicate over IP: The first approach is the implementation of IP communication and IP packet handling on the device. This approach is suitable for computational powerful devices, which have the resources to integrate the IP communication into their standard functionality. The second approach is to use a translator from the original communication protocol to IP. This concept is suitable for restricted devices, where the implementation of an IP stack is out of scope for the resources of these devices. As passive RFID tags are restricted devices, we follow the second approach. Before we describe the communication concept, we give a short overview of previous work done on integration of RFID tags into the Internet.

5.1.1 Previous Work

As already mentioned, the EPCglobal network was the first approach towards integration of RFID technology into the traditional Internet. In [Eng02], Engels compared the EPC identification scheme with the IP address scheme. He concluded that both schemes are similar in structure, but with essential differences: IP addresses cannot serve as unique identifiers, EPCs cannot be used for routing. In 2008, Yao-Chung et al. proposed a so-called IPv6-EPC Bridge

Mechanism [CYCSM08]. Here, each EPCglobal network is connected to the IP network via RFIPv6 gateways. Nodes outside the EPCglobal network can find information on the tags using the anycast mechanism of IPv6. IP addresses are created from the EPC and the subnet address of the EPC Information Service (EPCIS) server.

In 2007, Sang-Do et al. presented a concept on the use of EPCs to create conventional IPv6 addresses [SDMKHJ07]. In this concept, the network prefix of the IPv6 address is replaced by a 64-bit EPC. The address generation is done by the reader, where the tag is present, which transmits it to the (EPCIS) server. The EPCIS service is defined for storing and retrieving information about particular tags on the EPCglobal network. In the system proposed by Sang-Do et al. the current IP address of a tag is stored as additional information. If a node on the Internet wants to retrieve data from the tag, the EPCIS can request the tag information in real-time using the stored IP address. This scheme provides some useful concepts on network communication with RFID tags (e.g., address mapping, central server as relay for the tags). Therefore, we adapted some of the ideas in this work for the use in the proposed communication protocol.

Most published work mainly focuses on the EPC standards and the EPCglobal network. In our work we have a more general approach, which can be used for all types of RFID standards. Furthermore, we do not limit our communication concept to the EPCglobal network, but use the standard communication protocol on the Internet to connect passive RFID tags. In the proposed approach, RFID tags can be treated as mobile nodes in the IPv6 network. We use concepts from the Mobile IPv6 (MIPv6) scheme to integrate these tags into the IP network. In the next section we give an overview of MIPv6 concepts and describe how they can be used to build MIPv6-enabled RFID tags.

5.2 Mobile IPv6

The current standard protocol for Internet communication is the *Internet Protocol Version 4* (IPv4). As the address space of IPv4 is 32 bits wide, it can be predicted that the number of Internet-enabled devices will soon exceed the number of available IP addresses. Therefore, the successor of IPv4, the *Internet Protocol Version 6* (IPv6) was developed. The extension of the address space to 128 bits is the most important change from IPv4 to IPv6. With that address space about 1500 addresses per square foot of the earth's surface can be issued. Thus, IPv6 can be a basis for an *Internet of Things*, where a huge amount of mobile devices want to communicate and need IP addresses. Additionally, the IPv6 was provided with new functionality and more flexibility. In the following, we describe the basic properties of IPv6, including the mobility concept, that are required for the development of remote communication with passive RFID tags on the Internet.

5.2.1 MIPv6 Addresses and Packets

IPv6 is specified in RFC 2640 [DH98]. IPv6 addresses hold 128 bits which are separated into 16-bit blocks. Each block is represented by a hexadecimal number consisting of 4 digits, e.g., `2354:1493:2984:AF53:0001:0000:3AFB:103B`. If subnets are used, like in mobile IPv6, the first 64 bits of the address are defined as the *Address Prefix*. The address is divided into the network part, which corresponds to the address prefix, and the host part. The network part contains the subnet identifier, which uniquely identifies a subnet where a node is present. The last 64 bits, the host part, represents a unique identifier of the device.

The address types that are defined in IPv6 are unicast, multicast, and anycast addresses. Unicast addresses exactly identify one particular destination. A multicast address is shared by a group of nodes. Multicast addresses are used to reach more than one destinations in parallel by only specifying one destination address. An anycast address is also shared by a group of nodes, but here the packet is only sent to one node which holds the address. This is, for example, the next reachable node with the specified address. Starting from the first node, the packet is spread to all other members of the group. This concept is useful for mobile nodes, as not all members of the group must be online at the same time.

In order to provide the Mobile IPv6 (MIPv6) functionality a mobile node is identified by a EUI-64 address (Extended Unique Identifier). This address consist of 64 bits and is derived from the MAC (*Media Access Control*) address of the mobile node. The MAC address is a physical LAN address consisting of a Company ID (24 bits) and a Manufacturer ID (24 bits). These two values are separated and two bytes (0xFF, 0xFE) are inserted in the middle, which leads to a 64-bit address. The mobile IPv6 address is composed of the subnet prefix of the node's home subnet and the derived address from the MAC. This process is called *stateless address (auto)configuration* and specified in RFC 4862 [TNJ07] and RFC 4861 [NNSS07]. Figure 5.1 shows the structure of an MIPv6 address. An IPv6 packet consists of header and payload. The header has a static part

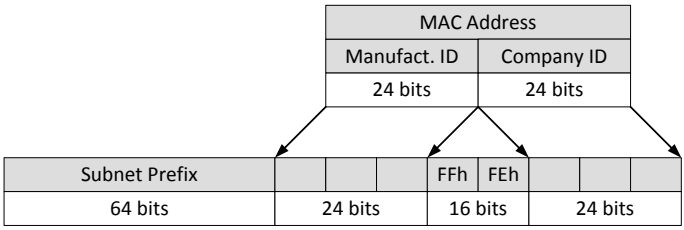


Figure 5.1: Structure of an MIPv6 Address

of 40 bytes, which we call the basic header, and can be extended by optional headers, called extension headers. The basic IPv6 header consists of

- a version field (4 bits) which indicates the version of the IP protocol,

- a traffic-class field (8 bits) where a priority of transported packages can be defined,
- a flow-label field (20 bits) to indicated that some packages can be routed directly over an end-to-end connection,
- a payload-length field (16 bits) where the number of following payload bytes is indicated,
- a next-header field (8 bits) which indicates the header type following the basic header, for example, TCP header, or UDP header, or an extension header,
- a hop-limit field (8 bits) which indicates the maximum number of routers a package can pass until it is deleted,
- and a source and destination address field with 128 bits each.

Various extension headers, following the basic header, are possible, namely: Hop-by-Hop Options Header, Routing Header, Fragment Header, Destination Options Header, Authentication Header, Encapsulation Security Payload, and Mobility Header. Mobile IPv6 packets contain the *Mobility Header* and the *Routing Header*. For the proposed concept, the *Mobility Header*, *Routing Header* and the *Fragment Header* are of interest. The *Mobility Header* contains various messages to provide mobility of nodes (e.g., binding messages). We describe these mobility concepts in the next section. The *Routing Header* is used to control point-to-point communication between a mobile node and a correspondent node. With the *Fragment Header* an IP packet can be separated into various parts and can be composed again at the destination node. The *Authentication Header* and the *Encapsulation Security Payload* are useful for securing an IP connection. We explain the use of these headers in Chapter 6, when introducing IPSec basics.

5.2.2 Mobile IPv6 Concepts

The IPv6 standard provides concepts on how to handle mobile nodes. A node is called mobile if it is able to move within the network from one subnet to another. Different routing and detection mechanisms have to be provided than for "static" nodes. The MIPv6 concept is described in [JPA04]. A mobile node has a home subnet where the so-called *Home Agent* (HA) of the node is located. The *Home Agent* always knows the last locations (IP addresses) where its mobile nodes were situated. As already mentioned, the *Home Address* of the mobile node is the subnet address of the *Home Agent* concatenated with the EUI-64 address of the node.

If a *Correspondent Node* (CN) wants to send a message to the mobile node, it uses the *Home Address* of the node as destination address. Thus, the IP packet first reaches the *Home Agent* as the subnet prefix of the *Home Address* is the subnet address of the HA. The HA holds the so-called *Binding Cache* where all

current IP addresses of its mobile nodes are stored. The *Home Agent* looks up the current IP address (*Care-of Address*) in its *Binding Cache* and relays the message to this address. If the *Corresponding Node* is not MIPv6-enabled, all communication (also the responses from the mobile node) is relayed over the *Home Agent*. If the *Correspondent Node* is MIPv6-enabled, a point-to-point communication can be established, where the *Correspondent Node* stores the new *Care-of Address* of the mobile node and uses it as destination address for further messages.

MIPv6 defines processes to manage the mobility of nodes on the Internet. Mobile nodes are able to connect to different MIPv6 routers over time. Each MIPv6 router manages a particular subnet. If the mobile node leaves the subnet, where it was connected, the following steps are carried out:

- **Subnet Discovery**: Each MIPv6 router periodically sends a *Router Advertisement*, where the subnet prefix is contained. A mobile node listens to the *Router Advertisements* and if it receives two different prefixes subsequently, the node realizes that the subnets had changed. This procedure is called *Neighbor Discovery*.

- **Generation of new Address**: The IP address of a mobile node changes by moving from one subnet to another. The current IP address is composed by the current subnet prefix and the EUI-64 address of the node. The current IP address is called *Care-of Address* (CoA).

- **Discovery of *Home Agent***: The mobile node has to know the IP address of its *Home Agent* in order to communicate with it. This address can be manually configured for each mobile node. If the *Home Agent* address changes, the MIPv6 protocol provides a *Home Agent Address Discovery* service to find out the new address.

- **Setup of Security Association** (optional): If communication between the mobile node and the *Home Agent* is to be secured, IPSec can be used. For this purpose, mobile node and *Home Agent* have to agree on the security algorithms and protocols used in the communication. This agreement is called *Security Association* (SA).

- **Home Agent Binding**: The mobile node sends its *Care-of Address* to the *Home Agent* using a *Binding Update* message. The *Home Agent* stores the new IP address in its *Binding Cache*. Here the current IP addresses of all mobile nodes managed by the HA are stored. The *Home Agent* confirms the receipt of the new *Care-of Address* with a *Binding Acknowledge* message.

- **Correspondent Node Binding** (optional): If a point-to-point communication has been established between a *Correspondent Node* and the mobile node before the subnet changed, the node has to update this binding, i.e., it must send its new CoA to the *Correspondent Node*.

The most important steps in this procedure are the generation of new *Care-of Addresses* and the home-agent binding. Providing these features, a *Home Agent* can establish communication between each CN and the mobile node. Based on the described mechanisms, we have developed a system to connect passive RFID tags to the *Internet of Things*. We describe this system in detail in the next section.

5.3 Connecting RFID Tags via MIPv6

In this section we address two points which have to be considered when integrating RFID tags into a network: Addressing of the tags and communication principles. We discuss two approaches how IP addresses can be assigned to tags in Section 5.3.1. In the following Section 5.3.2 we show how Mobile-IPv6 concepts can be used for interactive communication with tags. As in Mobile IPv6, we work with a *Home Agent* which manages IP addresses and information about tags.

5.3.1 Addressing RFID Tags on the *Internet of Things*

In order to contact tags over an IP network it is necessary that the tags can be uniquely addressed. Like every other mobile node on the IPv6 network, they must hold an MIPv6 address. We have already described the structure of an MIPv6 address in Section 5.2.1. In principle, two approaches can be used to address tags within an IP network: The unique identifiers of the tags can be mapped to an MIPv6 address, or IP addresses can be assigned randomly to the tags. In the following we discuss these two approaches.

Address Mapping

Deriving the *Home Address* of a tag from the tag identifier, no additional communication is needed for a *Correspondent Node* to find out the tag's IP address. In this approach, we assume that the *Correspondent Node* knows the identifier of the tag and can therefore derive its *Home Address*. As different RFID communications standards exist (e.g., [Int00], [Int04b], [Int04c], [Int04a], or [EPC03]), there is no common structure for unique identifier of tags. Even within one standard, there can exist different types of identifiers with different structures. This means that, a general approach to map tag identifiers to MIPv6 addresses does not work. For each communication standard a specific addressing concept would have to be developed.

In the following, we give an example how mapping can be done for a GID-96 identifier of EPC tags: A GID-96 identifier consists of 96 bits, which are arranged in header, serial number, an object class, and a manager number. Figure 5.2 shows the structure of a GID-96 identifier. The header consists of 8 bits. It is followed by a 28-bit *General Manager Number*. This number is an unique identifier for the object manager, which is responsible for the assignment

of the serial-number and object-class fields. The object class is encoded as a 24-bit value. This value must be unique within the object-manager domain and identifies the object type. The 36-bit serial number is unique within an object class and uniquely identifies a particular entity of an object. For the GID-96

Header	Manager #	Obj. Class	Serial #
8 bits	28 bits	24 bits	36 bits

Figure 5.2: Structure of a GID-96 Identifier

identifier, 96 bits have to be mapped to the 128 bits of an MIPv6 address. As we have described in Section 5.2.1, the MIPv6 address consists of the subnet prefix and the EUI-64 address. The EUI-64 address is derived from the MAC address of the item and is unique for the device. The GID-96 identifier holds 28 bits to identify the company, which manages the tags (object manager). This *General Manager Number* can be used as part of the subnet prefix of the *Home Agent*. The object-class and serial-number fields are used to uniquely identify the item and can be mapped to the last 64 bits of the MIPv6 address which are item-related. In Figure 5.3 we show the address mapping from a GID-96 identifier to an MIPv6 address. The *General Manager Number* consists of 28

<table>
<tr><td colspan="2">Subnet Prefix (64 bits)</td><td colspan="3">Tag Identifier (64 bits)</td></tr>
<tr><td>Manager #</td><td rowspan="2">36 bits</td><td>Obj. Class</td><td>Serial #</td><td rowspan="2">4 bits</td></tr>
<tr><td>28 bits</td><td>24 bits</td><td>36 bits</td></tr>
</table>

Figure 5.3: Address Mapping from GID-96 Identifier to MIPv6 Address

bits, whereas the subnet prefix is decoded with 64 bits. Therefore 36 bits for the subnet prefix cannot be derived from the GID-96 identifier and have to be assigned randomly. If the subnet prefix depends exclusively on the *General Manager Number*, the available address space used to identify the *Home Agent* of a tag is reduced to 28 bits. Another issue is the transfer of ownership of a tag. We consider the tags to change their manager during lifetime. In case of an ownership transfer, the new *Home Agent* address has to be encoded into the unique identifier. This means that, the tag identifier has to be modified in this case.

For the item-related part of the MIPv6 address, 60 bits from the GID-96 can be used (serial number and object class). In this case, 60 bits are mapped to the 64-bit host part of the *Home Address*. The leads to a reduction of the item-related address space to 60 bits, which is not as drastic as for the subnet part. Only 4 bits have to be randomly assigned. In case of an ownership transfer, collisions can occur: The tag identifier is unique within the *Home Agent* domain, but this is not guaranteed for a new *Home Agent* domain.

Random Assignment

In difference to the address-mapping approach previously described, MIPv6 addresses could be randomly assigned to a tag at the issuing process. In this process, the unique identifier is assigned to the tag, too. We assume that the tag manager, which issues the tag, provides the *Home Agent* service for this tag. Thus, the tag manager can, besides the ID, assign a unique *Home Address* to the tag. The emph{Home Address conforms to the MIPv6 structure: The first 64 bits of the address consist of the subnet prefix of the *Home Agent* (= tag manager). The last 64 bits identify the tag uniquely within the *Home-Agent* domain. The *Home Address* of the tag is stored in the *Binding Cache* of the *Home Agent* as well as in the tag's memory.

If IP addresses are assigned randomly, the establishment of a look-up service is useful. Issuing a tag, the *Home Agent* registers the tag at the look-up server by sending the tag identifier and the *Home Address*. If any *Correspondent Node* (CN) wants to contact one particular tag it has to know the unique identifier. The CN can first contact the *Home-Address Look-Up Service* to derive the *Home Address* of the tag. In the following, the *Correspondent Node* can use the *Home Address* to contact the tag. In the random-assignment approach, a transfer of ownership can be easily performed. When a tag is assigned to a new owner and therefore to a new *Home Agent* domain, the tag's *Home Address* is changed as needed, and updated in the tag's memory as well as in the database of the look-up service.

5.3.2 Communication Principle

In the proposed communication scenario, four parties are involved: The tag, the reader, the *Home Agent* (HA) and the *Correspondent Node* (CN). The following prerequisites have to be met in order to set up remote communication with passive RFID tags over an IP network.

- The tag holds an unique MIPv6 address. The address assignment is discussed in Section 5.3.1.
- The *Home Agent* manages the tag information and stores the current IP address of the tag (*Care-of Address*) in the *Binding Cache*. Furthermore, the *Home Agent* processes all IPv6 packets addressed for tags that are managed by the HA.
- The reader is connected to the Internet and acts as an MIPv6 router and translator for the tags in the field. The reader manages the communication between tags and the network and translates the received MIPv6 packets into communication frames according to the used RFID communication standard, and vice versa.

Communication with tags works as follows: A *Correspondent Node* that wants to talk to an RFID tag sends an IPv6 packet, addressed with *Home Address* of the tag, over the network. The packet is first routed to the *Home Agent* as the

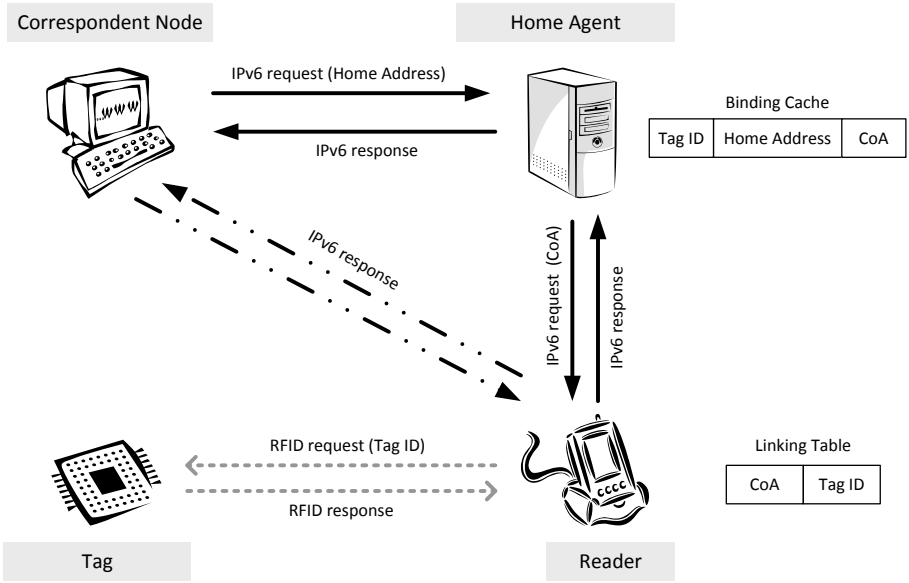


Figure 5.4: Communication Principle for MIPv6-Enabled RFID Tags

subnet prefix of the *Home Address* represents the *Home Agent's* subnet address. The *Home Agent* looks up the current IP address (*Care-of Address*, CoA) of the tag in the *Binding Cache*. The HA exchanges the destination address of the packet with the CoA. If the *Correspondent Node* has no MIPv6 capabilities, the *Home Agent* also exchanges the source address of the packet with its own IP address. Then the modified packet is sent again over the network.

As the destination address of the modified IPv6 packet is the current location of the tag, the packet is routed to the reader where the tag is present. The reader translates the packet into an RFID frame and sends the frame to the tag with the *Home Address* corresponding to the destination address. The tag sends an RFID response frame, which the reader re-translates into an IPv6 packet. This packet is sent back to the source address of the received packet. If the *Correspondent Node* has MIPv6 capabilities, the packet is returned to the CN, which can establish a binding and a point-to-point communication with the tag. If this is not the case, the response is sent to the *Home Agent*, which also serves as relay for all further communication. Figure 5.4 shows an overview of the proposed system.

In Section 5.2.2, we have described how mobility of nodes can be handled in IPv6. In the following, we discuss how the described concepts can be used for RFID tags, moving from one reader to another. Moreover, we can simplify the required steps when using RFID tags and shift most of the protocol complexity to the reader. The main difference to MIPv6 is, that not the mobile node (the tag, respectively) communicates with the *Home Agent*, but the reader does.

- **Subnet Discovery**(not required): This feature is not required in our concept. The tag does not care about its current location. If a tag enters a reader field, the tag indicates to the reader that it should be treated as mobile node. This can be done, for example, by modifying the response header during the inventory process. An inventory procedure is performed by every standard RFID reader. If the reader supports MIPv6 functionality it is able to handle this response header in an appropriate way. The tag does not have to be aware of this functionality.

- **Generation of new Address**: The reader generates the *Care-of Address* (CoA) of the tag. It uses its own subnet prefix for the first 64 bits of the CoA, the last 64 bits of the CoA are taken from the tag's *Home Address*. The tag itself is not aware of its new CoA. However, this is not necessary, as all network communication is managed by the reader.

- **Discovery of *Home Agent***: The reader sends a request to the tag to reveal its *Home Address* stored in the tag's memory. From this address, the subnet address of the *Home Agent* can be extracted. It is possible that the IP address of the *Home Agent* changes during the time the tag is not online. In this case, the reader receives an error response when sending a packet to the old IP address of the HA. The new address of the *Home Agent* has to be derived. This can be done by using the *Home-Address Look-Up Service*, described in Section 5.3.1. In this scenario the reader sends the unique ID of the tag to the look-up service and receives the new *Home Address* of the tag. The *Home Address* is also to be updated in the tag's memory. This can be done remotely by the *Home Agent* after an authentication procedure.

- **Setup of Security Association** (optional): As reader and *Home Agent* are participants on the IP network, they can establish a secure communication with the standard IPSec protocol. The IPSec concept is described in Chapter 6.

- **Home Agent Binding**: The reader sends the *Care-of Address* of the tag to the *Home Agent*. The HA stores this new address in its *Binding Cache*. In the *Binding Cache* the current IP addresses of all mobile nodes managed by the *Home Agent* are stored. The *Home Agent* confirms the receipt of the new CoA with a *Binding Acknowledge* message. The reader stores the *Care-of Address* and the tag identifier in a database, which we call *Linking Table*, in order to relay the received IPv6 packets to the correct tags. Figure 5.5 shows the updating of the tag's IP address.

- **Correspondent Node Binding** (not possible): Although a point-to-point communication can be established between a *Correspondent Node* (CN) and the RFID tag, corresponding node binding is not possible in this

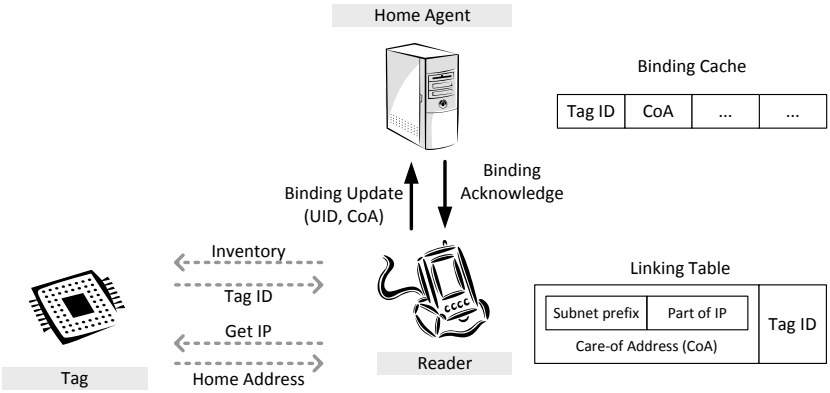


Figure 5.5: Home Agent Binding for MIPv6-Enabled Tags

case. The tags itself are not aware of the *Correspondent Nodes* they communicate with. The reader, where the tag is currently located, manages the connection and knows the IP addresses of the *Correspondent Nodes*. As the reader changes when the tag moves and the tag does not store the CN addresses, the new reader cannot be aware of former bindings of the tag. Thus, this step is omitted in the proposed communication protocol.

Using the indicated communication protocol, *Correspondent Nodes* can communicate with passive RFID tags via IPv6. The *Home Agent* service is an essential part of the system. Therefore, we suggest to use multiple *Home Agent* servers with an anycast address to provide this service. Using anycast addresses, several servers can hold the same IP address. Messages to the IP address can be received and handled by each of the servers. If necessary, the server that received the message forwards the packet to the other servers with the same anycast address.

In the proposed system this anycast addresses can be used as follows. For communication between *Correspondent Node* and tag, IPv6 packets are forwarded to the destination address. The first *Home Agent* server that receives the packet processes it. A forwarding of the communication packets to the other HA servers is not necessary, as no information about the communication is stored on the *Home Agent*. For address-updating messages, the *Home Agent* receiving the *Home Agent Binding* message forwards the updating information to all other servers, until all *Binding Caches* are up to date. In this way, the HA service is also available if one ore more servers are down.

The difference when dealing with passive RFID tags on the IPv6 network to conventional mobile nodes is the implementation of the IPv6 stack. In the proposed concept, RFID tags do not implement the IPv6 themselves, but use the reader as gateway to the network. All communication is translated and routed by the reader. In the next section we work out the capabilities RFID components must provide to execute the proposed communication concept.

5.3.3 Requirements for RFID Components

In the proposed system, readers act as translators and routers for the RFID tags. Therefore, RFID readers must be able to receive, process, and send IPv6 packets. Reader devices are in general connected to a computer, the so-called host, that executes the reader application, controls the reader, and processes the data received from the tags. Most computers are already connected to the Internet and have enough computational power to implement the IPv6 protocol. The hosts provide the IPv6 functionality for the readers. Furthermore, they offer their computational power for implementation of the translation and routing tasks.

The *Neighbor Discovery* functionality is used in MIPv6 for mobile nodes to check whether the subnet has changed. If a change of the subnet occurs, a new *Home Agent Binding* is initiated, i.e., the new IPv6 address of the mobile node is communicated to the *Home Agent*. In MIPv6 the mobile nodes themselves execute this binding, whereas in the proposed concept only the reader is involved in the communication with the *Home Agent*. The RFID tag itself is not aware of neither the current subnet nor of its current IP address. Therefore, the tag is not able to notice if the subnet has changed to initiate a *Home Agent Binding*. This function is taken over by the readers: A standard RFID reader continuously executes an inventory procedure to register all tags that are present in its radio field. In this way, the reader gets to know if a new tag has entered the field. An MIPv6-enabled tag responds to an inventory request with a modified inventory response, which means that a flag in the header indicates the MIPv6 capability. If such a tag enters the reader field, the reader has to find out the *Home Address* of the tag, generates a new *Care-of Address*, performs a *Home Agent Binding*, and stores the link between the tag identifier and the *Care-of Address* in the *Linking Table*.

The reader acts as a translator from IPv6 to the RFID communication standard required to communicate with the tag. The reader device looks up the destination address of the received IPv6 packet in its *Linking Table* and gets the ID of the addressed tag. The IPv6-packet payload is translated into a valid RFID frame and sent as an RFID request to the tag with the corresponding ID. In order to manage the communication with the tag, the reader stores the source address of the received IPv6 packet. Afterwards, this address is required to build up the response packet for the *Correspondent Node* or the *Home Agent*. The reader converts the tag response into an IPv6 packet and sends it back to the source of the received packet.

Translation of the IPv6 payload into an RFID frame and vice versa can be complicated. In the most simple case, the IPv6 packet holds the RFID-frame bytes as payload which can be directly sent to the tag. A different approach encodes the reader commands in the IPv6 packet payload, which can then be carried out by the reader. In these cases, the *Correspondent Node* has to be aware of the RFID communication standard to use for the destination tag. In order to provide a more general approach, we suggest to define a public IPv6-command set for RFID tags, which can be looked up on the Internet, that

can be easily translated into various RFID communication standards. The tag managers can decide which of these commands are supported by their tags.

RFID tags do not require too much additional functionality to conform to the proposed concept. MIPv6-enabled tags have to indicate, that readers should do a *Home Agent Binding*, when entering the field. This can be done with a flag in the inventory response header. They need an extra memory to store their *Home Address* (128 bits), and the capability to send and update this address. This can be done by defining two additional commands, for example, *GetIP* and *ChangeIP*. The complexity of the proposed approach is mainly handled by the reader. This is only true for standard communication, i.e., a *Corresponding Node* wants to send data to the tag or receive data from the tag without further functionality like security. Secure connections based on IPSec can also be established with RFID tags, although much more functionality is required on the tag in this case. We describe how to establish a secure end-to-end connection between a *Correspondent Node* and an RFID tag in Chapter 6.

In the most simple scenario of the proposed communication concept, the *Correspondent Node* triggers communication and the destination tag is always reachable ("online"). Some applications may require the tag to trigger communication, or they have to handle communication with "offline" tags. We describe how to deal with these scenarios in the next section.

5.3.4 Advanced Features for MIPv6-Enabled Tags

In applications that require the tags to report some actions to a server, or to request information from another node on the network, tag-triggered communication has to be provided. This means that, the tag indicates that it wants to establish a remote connection. This feature is not provided for standard RFID communication in which always the reader initiates the communication protocol. The tag indicates to the reader that it wants to establish an IPv6 communication by using a particular flag in the inventory response header. As the reader continuously polls for new tags in its radio field, the tag has the opportunity to send its inventory response during this polling.

Each RFID communication standard provides requests to exclude a tag from the inventory process, for instance, if required tag data have already been received. In this case, the tag has no possibility to indicate its communication requirement to the reader via the inventory response. Therefore, we propose to modify the standard functionality of advanced RFID tags: If an event occurs (e.g., a sensor measurement exceeds a limit) that triggers the tag to start remote communication, the tag can wake up from its "excluded" state and responds to the next inventory request. If a reader, that supports MIPv6-enabled tags, receives an inventory-response header where the communication flag is set, it sends a request asking for more information on the tag's communication needs. The tag sends data required to build an IPv6 packet that is to be sent to the *Correspondent Node*. These data has to consist at least of the destination address and the payload of the packet.

Another issue that has to be considered is the offline status of tags. If tags

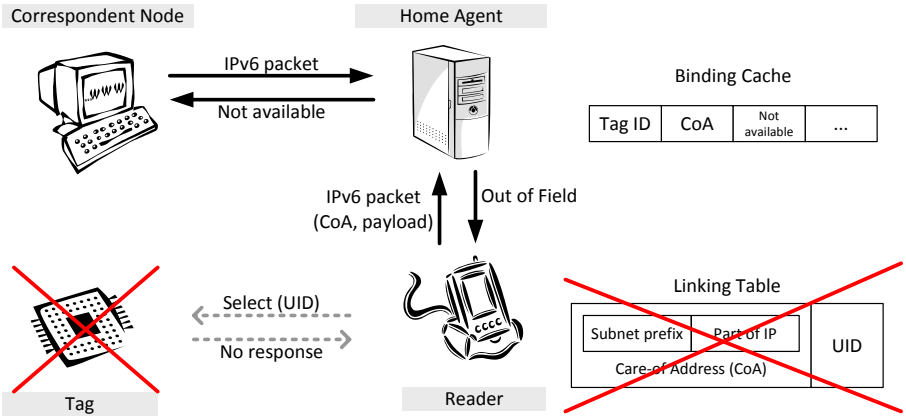


Figure 5.6: Offline Scenario for MIPv6-Enabled Tags

move, they are often not located in a reader field, which means that they cannot communicate. In the following, we describe how communication with offline tags can be established, anyway. If a *Correspondent Node* wants to contact a tag, the *Home Agent* relays the IPv6 packet to the last known IP address of the tag. If the reader receives a packet for a tag that is no longer present in its radio field, the reader sends an *Out-Of-Field* message to the *Home Agent*. The reader deletes the tag from its *Linking Table*. The *Home Agent* can update the *Binding Cache* and mark the tag as not available. It sends a *Not-Available* message back to the *Correspondent Node*.

Using extension headers of the IPv6 packet, the *Correspondent Node* can specify what should happen to the packet if the addressed tag is offline. If a prompt response to the packet is necessary, the packet is deleted after the *Home Agent* has got to know that the tag is not available. If a response to the packet can be useful within a certain time, the *Correspondent Node* can define a time-out for the packet. The *Home Agent* stores this packet for delayed transmission. If the *Home Agent* receives a new *Care-of Address* of the addressed tag before the specified time-out is reached, it re-transmits the packet to the tag. The further protocol is performed as in the standard procedure. Figure 5.6 shows the offline scenario for MIPv6-enabled tags.

5.4 Conclusion

In Chapter 5 we describe a system how to integrate passive RFID tags into the *Internet of Things*. The system is based on the mobility concept of IPv6 that deals with mobile nodes that can move through different subnets. We map this mobility concept to RFID tags, which also move through different reader fields. The reader is used as translator and router of IPv6 packets for the tags. A *Home Agent* manages the communication with the tags: If any *Correspondent*

Node wants to send a packet to a tag, it contacts the *Home Agent*, which holds the current IPv6 address of the tag in its *Binding Cache*. Then the packet is transmitted to this IPv6 address by the HA. If a tag enters a new reader field, the IPv6 address in the *Binding Cache* is updated. In this way, the *Home Agent* always knows how to contact the tag and offers a relay service for any other node on the net.

As RFID tags are constrained devices, the handling of the IPv6 communication is mainly performed by the reader. If a tag is not located in a reader field, the tag is not able to communicate. As RFID applications often require the tags to cover long distances from one reader to another, they are often offline. Therefore, we discuss strategies to establish offline communication with tags, where packets can be stored by the *Home Agent* and handled with a certain time delay. With the proposed system a two-way communication between any *Correspondent Node* on the Internet and passive RFID tags is established.

Communication with tags via IPv6 is a useful feature for RFID applications. However, most applications will profit from the possibility to secure online communication with RFID tags. Security is an essential feature, in particular for an open network like the Internet. As RFID tags in the proposed concept do not execute the IPv6 protocol itself, standard security measures for IPv6 connections, like the IPSec protocol, are not fully applicable. In the next chapter we show how to modify IPSec mechanisms in order to build a security layer for communication with MIPv6-enabled RFID tags on the Internet.

6

Security Layer for MIPv6-Enabled Tags

Security is an indispensable feature for quite a number of RFID applications. If applications require online access to RFID tags on the Internet, the establishment of a secure communication channel is essential. We learned from the traditional Internet, that protection of information is an important feature for applications. Information leakage can cause severe financial damage. Furthermore, the public acceptance of a technology depends on its reliability and security. Obviously, the *Internet of Things* will need secure communication mechanisms as well. In this chapter we present a security layer for MIPv6-enabled tags connected to the Internet.

The presented security layer is built upon the communication concept described in Chapter 5. In this concept, the IPv6 connection is set up by the reader, which manages the further communication with the RFID tag. The connection between the reader and any other node on an IP network can be secured by standard mechanisms, defined in the IPSec standard [KS05]. In this case, the secure channel ends at the reader device. Communication with the tag has to be secured separately. A secure channel between a tag and a reader can only be established if the tag trusts the reader, i.e., the tag can be sure that the reader does not modify or leak packets. As a tag in our scenario cannot trust the reader per default, additional security mechanisms are necessary to establish a secure point-to-point connection between the tag and the *Correspondent Node*. Although IPSec cannot be implemented without modifications on an RFID tag, we use concepts from IPSec to provide authentication and confidentiality for the proposed communication protocol. In the following, we describe IPSec basics and how to use them in order to establish a secure IPv6 connection involving MIPv6-enabled RFID tags.

The chapter contains results of joint work together with Stefan Kraxberger.

Parts of the outcome have been published in the Journal of Security and Communication Networks, Special Issue on Protecting the Internet of Things [DK11].

6.1 IPSec Basics

Internet Protocol Security (IPSec) is a set of security mechanisms that provide confidentiality, authenticity, and integrity of an IP communication. The use of IPSec is optional for IPv4, but is mandatory for IPv6. IPSec was standardized by the *Internet Engineering Task Force* (IETF) in several documents. The general architecture is described in [KS05], which is extended by other RFCs (*Request for Comments*) specifying various cryptographic protocols and key management mechanisms that can be used in IPSec. In the following, we describe the basic principles of IPSec in order to identify mechanisms that can be used for securing communication with RFID tags over IPv6.

6.1.1 Security Association

IPSec is used to provide secure end-to-end communication between two nodes on an IP network. Before a secure connection can be established, the communicating partners have to agree on a set of algorithms and parameters that are used to encrypt and authenticate the exchanged messages. Such a set is called *Security Association* (SA). Each SA is uniquely defined by a *Security Parameters Index* (SPI), the destination IP address and the used IPsec protocol (AH or ESP, described later on).

Security associations are established using the *Internet Security Association and Key Management Protocol* (ISAKMP). ISAKMP can be used with different mechanisms such as pre-shared secrets or Internet Key Exchange (IKE) [HC98, Kau05]. In the proposed communication protocol, including passive RFID tags, a modification of IKE version 2 (IKEv2) is used for establishing a *Security Association*. We describe the IKEv2 protocol in Section 6.1.5.

Each node that wants to securely communicate on the network has to provide a *Security Association Database* (SADB) where *Security Associations* for particular connections are stored, i.e., this database holds information how a particular incoming packet must be handled and how an outgoing packet must be protected. The lifetime of an SA is stored in this database, too. Furthermore, The SADB contains information about the mode of operation used. Each host or gateway has its own *Security Policy Database* (SPD) which specifies on which type of packets IPSec protection should be applied, which should be bypassed, and which should be discarded.

6.1.2 Modes of Operation

Secure end-to-end communication can be provided to a pair of hosts, a gateway and a host, or to two gateways on an IP network. Hosts are the endpoints of a communication channel, whereas gateways are intermediate nodes that

provide routing functionality for other nodes. Two modes of operation can be distinguished for IPSec: The *Transport Mode* and the *Tunnel Mode*.

- **Transport Mode:** This mode is used for host-to-host communication. The IPSec header is inserted as extension header between the IP header and the payload. The IP header is not modified in this mode. As the IP header contains the source and destination address, routing can be done like for a regular IP packet. Only the payload of the IP packet is protected. Nevertheless, if used with an *Authentication Header*, described in the following, the source and destination address of the packet are included in the integrity check, so they cannot be modified without detection.

- **Tunnel Mode:** This mode can be used for gateway-to-gateway, host-to-gateway, or host-to-host communication. Here, the entire IP packet is authenticated and/or encrypted. The result is embedded into a new IP packet. The source and destination addresses in the new IP header can be different from the endpoints of the communication, as, for instance, only the gateway can be addressed. When the gateway receives the packet, it removes the encapsulation and receives the original IP packet which is then forwarded to the host corresponding to the destination address in the original IP header.

In each mode of operation, IPSec works with extension headers of an IP packet. We have described the structure of an IPv6 packet and the use of extension headers in Section 5.2.1. *Transport Mode* as well as *Tunnel Mode* can be used with *Authentication Header* (AH) and/or with *Encapsulating Security Payload* (ESP). We describe the use of these extension headers in the following.

6.1.3 Authentication Header

The *Authentication Header* (AH) is defined in [Ken05a]. It holds a sequence number, which is intended to prevent replay attacks, and the authentication data, which is in principle a checksum over the payload and some parts of the header. Since some of the IP-header fields are modified during transmission of the IP packet, not all header fields can be protected by using the AH. Authentication data is calculated using algorithms such as keyed *Message Authentication Codes*. These algorithms are based on symmetric encryption algorithms or on one-way hash functions. A keyed *Message Authentication Code* provides data integrity and authentication of the sender.

The *Authentication Header* is used as an extension header in the IPv6 standard. It contains the following fields: *Next Header*, *Payload Length*, *Security Parameters Index (SPI)*, *Sequence Number*, and *Integrity Check Value (ICV)*. The SPI is used to identify the *Security Association* of the sending node. The sequence number is used to protect the packet against replay attacks. The ICV contains a checksum for verifying data integrity. Figure 6.1 shows the use of the *Authentication Header* in *Transport Mode* and *Tunnel Mode*.

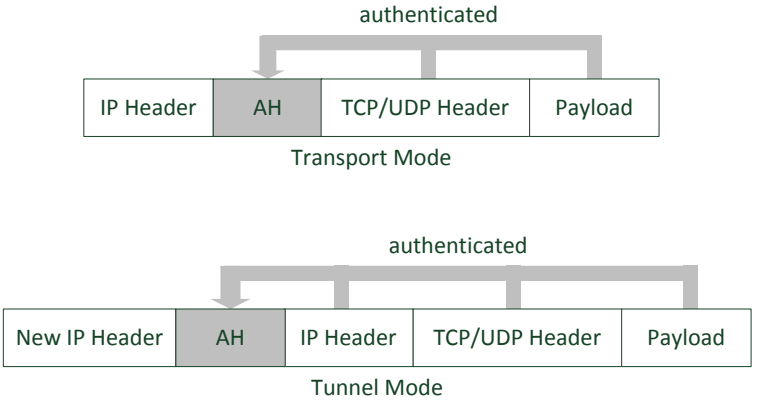


Figure 6.1: Usage of Authentication Header in IPSec

6.1.4 Encapsulating Security Payload

The *Encapsulating Security Payload* (ESP) is an extension header used in IPv6 packets. It is defined in [Ken05b] and can be used stand-alone or in addition to the *Authentication Header*. ESP provides confidentiality, data integrity and authentication of the sender. Nevertheless, it does not provide protection of the IP header. Only if used in *Tunnel Mode*, the IPv6 header is included in the protection. ESP is separated into two parts when used in an IPv6 packet: The

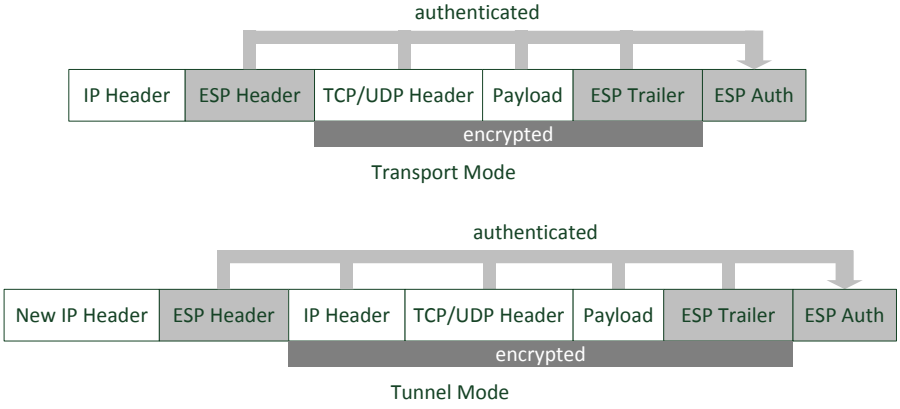


Figure 6.2: Usage of Encapsulating Security Payload in IPSec

ESP header and the ESP trailer. The ESP header consists of a *Security Parameters Index (SPI)*, used to identify the *Security Association*, and a *Sequence Number* for prevention of replay attacks. The ESP header is followed by the payload of the original IP packet. The ESP trailer is located after the payload and contains a *Padding* field, a *Padding Length* field, which contains padding

data that has to be used to provide a correct input-data length for the used encryption algorithm, and a *Next Header* field. At the end of the packet, an *Integrity Check Value (ICV)* is added which is used to provide data integrity of the IP packet. All fields starting from the ESP header and ending with the ESP trailer are encrypted, and thus confidentiality of the data is provided. Figure 6.2 shows the use of ESP in *Transport Mode* and *Tunnel Mode*.

6.1.5 Key Exchange and SA Agreement

A common key-establishment mechanism in IPv6 is the *Internet Key Exchange version 2* (IKEv2). This key-agreement protocol is defined in [KHNE10]. The actors in the scenario are the *Initiator*, which initiates the communication, and the *Responder*, which answers to the messages of the *Initiator*. The protocol is divided into two phases: During phase one, keys are exchanged using the *Diffie-Hellman* key-exchange algorithm [DH76] and a secure and authenticated channel is established. During phase two, *Security Associations* for other services, like IPSec, are negotiated. The IKEv2 protocol is shown in Figure 6.3. The *Initiator* starts phase one by sending an `IKE_SA_INIT` request. This mes-

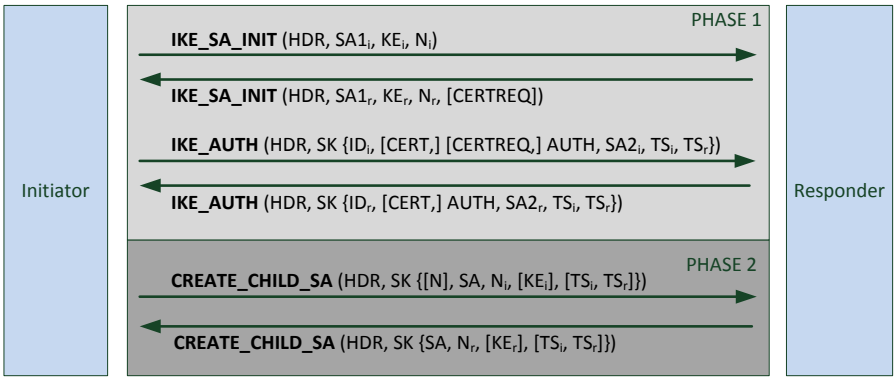


Figure 6.3: Key Exchange and Security Association Agreement in IPSec

sage basically consists of a suggestion for a *Security Association* ($SA1_i$), the public *Diffie-Hellman* value (KE_i) and a random number N_i. The header of all key exchange messages (HDR) contains the Security Parameter Indices (SPIs), version numbers, and various flags. $SA1_i$ contains all cryptographic algorithms supported by the *Initiator* for the key exchange procedure. The *Responder* answers the request with an `IKE_SA_INIT` response. The response contains the selected *Security Association* ($SA1_r$), which is a subset of cryptographic algorithms and protocol suggested in $SA1_i$, the public *Diffie-Hellman* value of the *Responder* (KE_r), and a random number N_r. Optionally, the *Responder* can add a certificate request ([CERTREQ]) to the `IKE_SA_INIT` response. From these two `IKE_SA_INIT` messages, *Initiator* and *Responder* calculate a session

key (SKEYSEED) for this session by using the following equation.

$$SKEYSEED = PRF(N_i|N_r, g^{S_x})$$

S_x denotes either be the secret of the *Initiator* (S_i) or the *Reponder* (S_r). Each of the two parties uses its own secret to calculate the session key. g can be derived from the public *Diffie-Hellman* values, which were exchanged before. PRF refers to a pseudo-random function. Details on the calculation of the session key can be found in [DH76], where the *Diffie-Hellman* key-exchange algorithm is described. From the session key several other keys are derived. Keys for encryption are denoted with SK_e, keys for authentication are denoted with SK_a. For each direction separate keys for authentication and encryption are computed, which results in four keys: SK_{ai} and SK_{ei} for the *Initiator*, and SK_{ar} and SK_{er} for the *Responder*. Furthermore, both parties derive a master secret (SK_d) from SKEYSEED.

After exchange of the `IKE_SA_INIT` messages, the *Initiator* sends an `IKE_AUTH` request. This message exchange is used for mutual authenticate of both parties. The message contains the identity of the *Initiator* (ID_i) and an authentication parameter (AUTH), which authenticates the identity of the *Initiator* and protects data integrity of the message. Furthermore, the message contains a suggestion for a *Security Association* for the next phase ($SA2_i$), and the traffic selectors TS_i and TS_r, which define what traffic has to be protected by this SA. The payload of the message is encrypted using the SK_{ei} key, derived from the session key.

The authentication parameter AUTH can be either generated using a pre-shared key derived from SKEYSEED, or by using the public key of the *Initiator*. Pre-shared keys do not authenticate the identity of the *Initiator*, i.e., the *Initiator* could use any identity in this message. The *Responser* can not be certain about the correctness of the identity parameter. Nevertheless, the AUTH parameter proves that the *Initiator* is the same party that has started the protocol, as only the *Initiator* and the *Responder* know the session key (and all other keys derived from it). The *Initiator* can also use its public key to generate the AUTH parameter. If a certificate has been requested by the *Responder* in the previous message, the certificate to this public key is supposed to be sent within the `IKE_AUTH` request. A certificate authenticates the identity of the *Initiator*, as the certificate links its identity to the public key. If the *Initiator* wants the *Responder* to send its public key as well, an optional certificate request ([CERTREQ]) is appended to the `IKE_AUTH` request.

The *Responder* answers the request with an `IKE_AUTH` response, which contains the identity of the *Responder* (ID_r) and the authentication parameter AUTH. As for the *Initiator*, the *Responder* can use either a pre-shared key or its public key to calculate the authentication parameter. If a certificate has been requested by the *Initiator*, the *Responder* appends its certificate to the message ([CERT]). The response also contains the selected SA for the next step ($SA2_r$), which is a subset from the suggested algorithms and protocols in $SA2_i$. Furthermore, the `IKE_AUTH` response holds the traffic selectors TS_i and TS_r.

The payload of the response is encrypted using the SK_{er} key. With the receipt of the `IKE_AUTH` response by the *Initiator*, phase one ends.

In phase two the *Security Association* used for IPSec is negotiated. A possible SA was already agreed upon in the `IKE_AUTH` messages. The parties can use either this SA or negotiate a new one. The *Initiator* starts phase two by sending a `CREATE_CHILD_SA` request. This request must contain an offer for a *Security Association* (either the previously agreed one or a new offer) and a random number N_i, which should be different from the random number used in phase one. Optionally, the request can contain a parameter N, which indicates that a new SA should be negotiated, and a new *Diffie-Hellman* value, which can be used to calculate new keys for IPSec. The traffic selectors TS_i and TS_r can also be contained in the message.

The *Responder* answers the request with a `CREATE_CHILD_SA` response, which contains the accepted SA and a random number N_r. If a new *Diffie-Hellman* value is required, this value is added as KE_r. If necessary, the message contains the traffic selectors TS_i and TS_r. After exchange of these messages, the keys for IPSec (for AH and ESP) are either derived from the master key SK_d previously calculated or by using the new *Diffie-Hellman* values. The random numbers are used in this calculation to refresh the key material.

After the IKEv2 protocol has been executed, a secure channel between the two involved nodes is established. IPSec uses the negotiated parameters to establish this secure connection. The essential parameters for this connection are the mode of operation (*Tunnel Mode* or *Transport Mode*), the selected extension header (*Authentication Header* or *Encapsulating Security Payload*), and the keys. The IKEv2 key-exchange mechanism is computational demanding. As RFID tags are computationally constrained, we researched on modifications that can be done to the IKEv2 protocol in order to provide the same functionality but with less computational effort. We discuss the results of this research in the next section, where we present a security layer for communication with MIPv6-enabled tags on the IPv6 network.

6.2 Secure Communication with RFID Tags

We have presented a communication concept on RFID tags connected to the Internet in Chapter 5. Online communication requires security measures, which is provided by the IPSec protocol for standard IPv6 communication. IPSec was not designed for constrained devices like passive RFID tags. Nevertheless, passive RFID tags are already able to perform complex calculations like cryptographic algorithms. We have given some examples of available primitives in Section 2.1.2. Furthermore, we have shown in Chapter 3 that secure applications can be built upon RFID technology. These findings lead to the conclusion that also a secure channel over the Internet can be established with passive RFID tags.

In this section we show how the IPSec protocol can be modified to be suitable for communication with MIPv6-Enabled tags. In Section 6.2.1 we define

the restrictions that have to be imposed on the IPSec protocol. Section 6.1.5 describes the modified key-exchange mechanism, which is followed by a discussion of the establishment of a secure end-to-end communication with passive RFID tags over IPv6.

6.2.1 Assumptions

Due to the limited resources of RFID tags we impose some constraints on the IPSec protocol. Assumptions have to be made, especially as regards traffic load, availability of keys, and functionality of the reader. We describe these assumptions in detail in the following.

IPv6 packet size

Standard IPv6 packets can have a size of up to 64 kBytes. It is also possible to create huge "Jumbo" packets by using the *Hop-By-Hop Options* extension header. We have discussed in Section 2.1.1 that state-of-the-art RFID tags can hold up to 64 kBytes of volatile memory. Thus, RFID tags will not be able to handle large IPv6 packets. In the proposed concept, a tag does not handle the IPv6 packets itself, but the readers translate the packets into RFID frames. Nevertheless, at least the payload of the IPv6 packet will be forwarded to the tag, which has to process these data.

The payload of huge packages exceeds a tag's memory and also processing capabilities. Therefore, we limit the size of data which is allowed to be sent from any node on the network to the tag. We do not define a fixed maximum size at this point, as the particular limit depends on the capabilities of the used tags and can change over time. The restriction on the IPv6 packet size is not a drastic one, as RFID applications are in general not data intensive for the tags.

Number of Connections

Due to memory and processing limitations of the RFID tag we limit the number of open IPSec connections to one. This means that only one *Correspondent Node* is able to communicate with a particular RFID tag at a time. If we would allow various numbers of connections, each connection has to be managed by the tag, maybe with different *Security Associations*. This could easily exceed the computational power and memory capabilities of a tag.

We also refuse connections from other nodes during the setup phase and impose a kind of minimum timeout interval. As soon as a connection from one *Correspondent Node* is accepted by the tag, no other node can connect to the tag during a certain amount of time. This amount of time should be correlated to the time used for the setup of a communication (key exchange, SA agreement) and the time used to exchange data. If we did not define such a timeout interval, it would be possible to break an existing connection during or directly after the SA agreement. In this case, there would be not enough

time to send application-related data to the tag. We want to avoid this case by imposing a timeout interval.

Traffic Policy

In order to reduce the computational effort, we apply the same policy to all network traffic. We assume that all network traffic is to be secured by the use of IPSec. In this way, we omit the management of the *Security Policy Database*. Furthermore, we limit the traffic policy to only one kind of protocol and one mode of operation, namely we use *Encapsulating Security Payload* in *Transport Mode*. We define the *Encapsulating Security Payload* as extension header as we want to provide authentication as well as confidentiality, which is not provided by using the *Authentication Header*. The *Transport Mode* is used as it provides a secure end-to-end communications between two hosts. Only the payload, not the IP header, is encrypted in this mode. In this way, routing and the handling of the IP header can be done as usual. In the proposed system, the reader takes over the IPv6 handling. Using *Transport Mode*, the reader is still able to provide the IPv6 communication for the tag. Therefore, the IP header is not sent to the tag. Thus, the communication frames are smaller and only the payload has to be processed by the tag.

The *Security Association* is a set of possible cryptographic algorithms for particular security services during an IP communication. As cryptographic algorithms are often computationally demanding, passive RFID tags will only provide a small set of them. Therefore, we define only one *Security Association* per tag. In this way we also omit the management of a *Security Association Database*, which reduces complexity again.

Certificates

Certificates can be used for the authentication process during the IKEv2 protocol. In principle, certificates are authenticated public keys. In public-key cryptography each communicating party holds a public/private key pair. The public key of each party can be signed by a *Certification Authority* (CA), which is a highly trusted party in the so-called *Public-Key Infrastructure* (PKI). In standard public-key infrastructures, each party stores its own certificate and sends it to the communicating counterpart when establishing a secure connection.

For the proposed communication concept, the standard use of certificates is modified. In order to save memory and computational effort on the tag, the certificates are managed by the *Home Agent*. In our case, the *Home Agent* takes over the role of the CA and stores the certificates as well. When issuing a tag, a public/private key pair is generated and stored on the tag. The public key of the tag is signed by the *Home Agent* and the resulting certificate is stored in the *Binding Cache* of the *Home Agent*. For a transfer of ownership, where the tag is transferred to a new tag manager, the new *Home Agent* can sign the public key again and save the new certificate in its *Binding Cache*. Figure 6.4

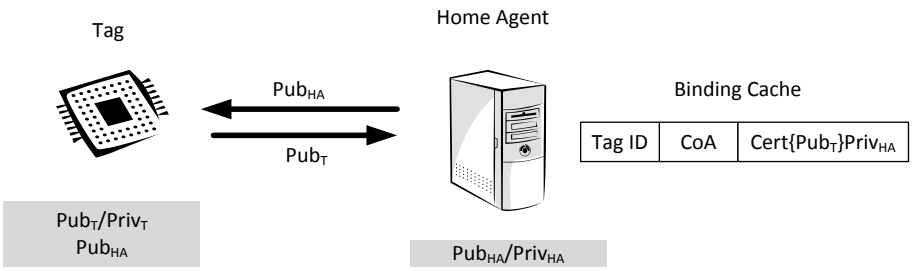


Figure 6.4: Issuing of a Tag Certificate by the Home Agent

shows the issuing of a tag certificate. $Priv_{HA}$ and Pub_{HA} denote the private and public key of the *Home Agent*, $Priv_T$ and Pub_T denote the private key and the public key of the tag. The certificate is denoted as $Cert\{Pub_T\}Priv_{HA}$.

A *Correspondent Node* which wants to send confidential messages to the tag or verify a signature generated by the tag needs the public key of the tag. If the tag is supposed to prove its identity, a certificate on this public key is necessary. As we want to reduce communication effort with the tags, the CN applies for the tag's certificate at the *Home Agent*. Figure 6.5 shows how the certificate is exchanged. *Pub* denotes a public key, *Priv* denotes a private key, and *Cert* denotes a certificate. The indices are defined as follows: CN for the *Correspondent Node*, T for the tag, and HA for the *Home Agent*. In order to

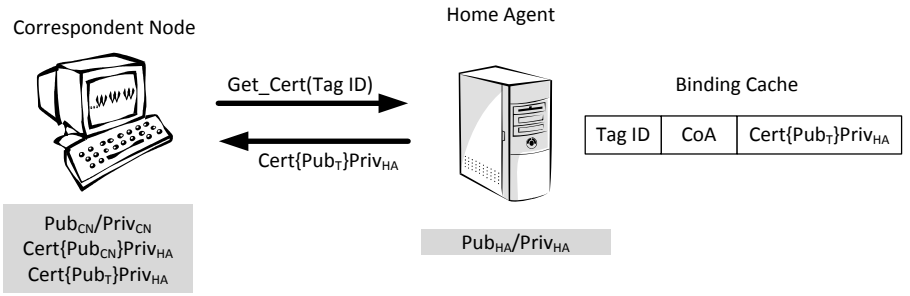


Figure 6.5: *Corresponding Node* Requesting the Tag Certificate

authenticate to the tag, the *Correspondent Node* needs a certificate from the *Home Agent* as well. In order to receive a certificate, the CN sends its public key to the *Home Agent*. Then, the HA can decide wether the node is allowed to talk to its tags or not. If the *Correspondent Node* is considered as a possible communication partner for the tags, the *Home Agent* signs the public key of the *Correspondent Node* and sends the resulting certificate back. A tag is able to verify the certificate of the CN by using the public key of the *Home Agent*. For the sake of simplicity, we propose to store the *Home Agent's* public key on the tag during the issuing phase of the tag.

If the *Correspondent Node* and the tag have verified each others certificate,

the public keys can be used for confidential and authenticated messaging. When using IPSec for the establishment of a secure IP connection, certificates can optionally be used during execution of the key-exchange protocol. In the next section we show how a modified version of IKEv2 can be used to exchange keys with passive RFID tags.

6.2.2 Key Agreement

The key-agreement mechanism for MIPv6-enabled RFID tags is based on IKEv2, described in Section 6.1.5. In order to unburden the tag from several resource-consuming tasks we modify the key-exchange protocol in some steps. In the following we describe the steps of the modified IKEv2 protocol in detail.

As in the IKEv2 protocol, the *Correspondent Node* is the *Initiator* and starts phase one of the key agreement by sending an `IKE_SA_INIT` request. This request contains the same values as in the original version of IKEv2: The *Security Association* ($SA1_i$), the public *Diffie-Hellman* value (KE_i) and a random number N_i. The message is first routed to the *Home Agent*. Here, the first modification to the IKEv2 protocol applies: The first protocol step is not handled by the tag, but by the *Home Agent*. The HA holds information about the *Security Association* provided for key exchange by the destination tag. In the proposed concept, only one SA for each tag is allowed, as described in Section 6.2.1. The *Home Agent* stores the SAs for key exchange for all its tags in the *Binding Cache*. As this *Security Association* is only transmitted in this first step, which is handled by the HA, there is no need to store it on the tag as well.

The *Home Agent* handles the first part of the key-agreement protocol in the following way: If the suggested *Security Association* $SA1_r$ contains no algorithms that are feasible for the tag, the *Home Agent* refuses the connection. If an agreement is possible, the *Home Agent* replaces the suggested $SA1_i$ by the single *Security Association* supported by the tag ($SA1_r$). Then the message is transmitted to the *Care-of Address* of the tag. The reader, where the tag is located, receives the IPv6 packet and translates it into an RFID frame (`IPsec_KE_Init`). The reader omits the $SA1_r$ parameter as the tag does not hold any data about the *Security Association* and cannot process this information.

The tag generates a response to the `IPsec_KE_Init` command. The response contains its public *Diffie-Hellman* value (KE_r) and a random number N_r. The tag can optionally request a certificate from the *Correspondent Node* by inserting a certificate request ([CERTREQ]). From these data, the reader builds an *IKE_SA_INIT* response. The reader inserts the $SA1_r$ parameter and sends the response back to the source address of the received packet (which can be either the IP address of the *Home Agent* or the *Correspondent Node*). Figure 6.6 shows the first step of the proposed key-exchange mechanism.

After exchange of the `IKE_SA_INIT` messages, the *Correspondent Node* as well as the tag calculate the SKEYSEED value, like described in Section 6.1.5. They derive the encryption and authentication keys SK_{ex} and SK_{ax} as well as the master key SK_d from the SKEYSEED value. All following messages are

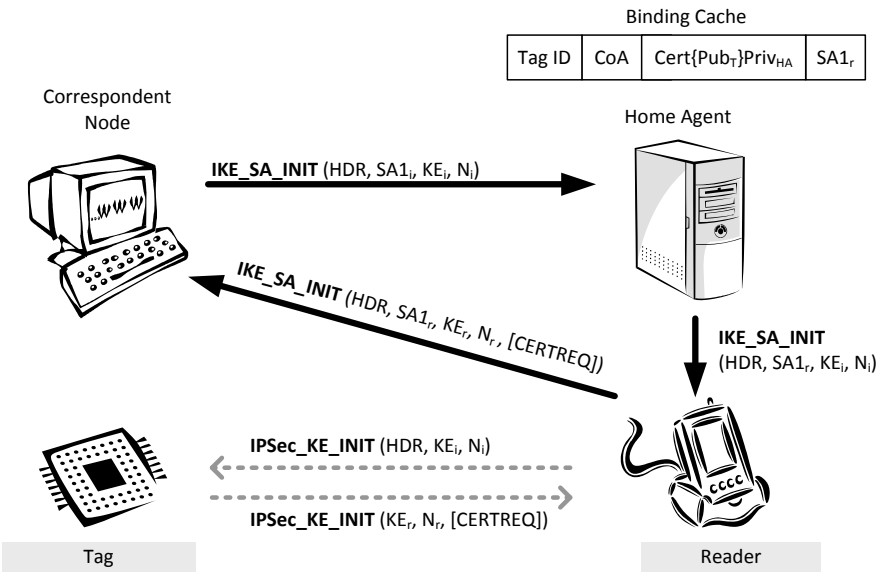


Figure 6.6: Key Exchange Initialization for RFID Tags

encrypted with the SK_{ex} and authenticated either with the SK_{ax} keys or by using public keys of the participants. If public keys are used, certificates can be requested to link the identity of the sender to its public key.

In the next step, the *Correspondent Node* sends an `IKE_AUTH` request containing its identity ID_i, its certificate [CERT] (if requested), an optional certificate request [CERTREQ], an authentication parameter AUTH, a suggestion for the IPSec *Security Association* $SA2_i$, and the traffic selectors TS_i and TS_r. Depending on the MIPv6 capability of the *Correspondent Node*, the message is sent to the CoA of the tag or sent via the *Home Agent*. For the sake of clarity we omit the HA in our further description as its task in the further steps is limited to relaying the messages and has no influence on the key-exchange protocol.

The reader translates the received `IKE_AUTH` request into an RFID frame (`IPsec_KE_Auth`). As the payload of this message is encrypted with SK_e, which the reader does not know, and the header is included in the authentication, the whole packet is transmitted to the tag. The tag can decrypt the payload and verify the authenticity of the sender and the integrity of the message. If a public key with certificate is used for authentication, the tag has to verify the validity of the certificate before verifying the AUTH parameter. If all verifications are successful, the tag has to check the *Security Association* $SA2_i$. If the *Security Association* for communication with IPSec, which is stored on the tag, for example, during the issuing process, is a subset of $SA2_i$, the tag accepts the key exchange.

The `IPsec_KE_Auth` response consists of a header and an encrypted and authenticated payload. The payload includes the identity of the tag ID_r, a

certificate link CERTLINK (if requested), an authentication parameter AUTH, the accepted *Security Association* $SA2_r$, and the traffic selectors TS_i and TS_r. We have already discussed the certificate handling of the tag in Section 6.2.1. The certificate of the tag is not stored on the tag itself, but in the *Binding Cache* of the *Home Agent*. If certificates are used for key exchange, the *Correspondent Node* has to request the certificate of the tag from the *Home Agent*. Instead of the certificate, the tag sends the link to the *Home Agent* to the *Correspondent Node*. Figure 6.7 shows the authentication step of the key exchange. After

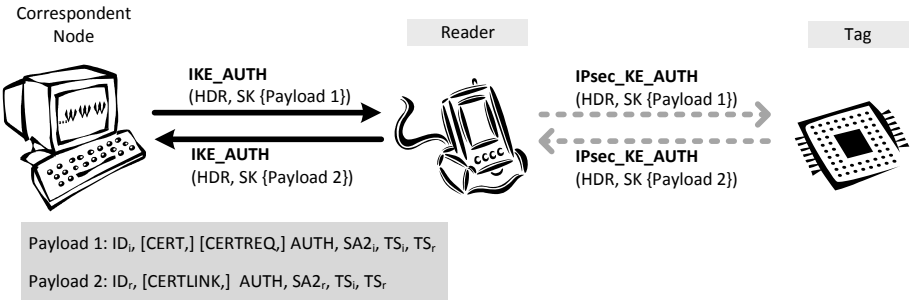


Figure 6.7: Key Exchange Authentication for RFID Tags

exchange of the `IKE_AUTH` messages, the key-exchange protocol enters the second phase. The *Correspondent Node* sends a `CREATE_CHILD_SA` request, which is used to set up the *Security Association* for IPSec. Although a SA for IPSec has already been negotiated in the previous step, the CN could request the negotiation of a new *Security Association* as described in Section 6.1.5. For the proposed key-exchange protocol with MIPv6-enabled tags this feature is omitted because the tag is computationally limited and only one SA for IPSec per tag is allowed. Thus, new key negotiation would result in the same SA than proposed in the authentication step. The `CREATE_CHILD_SA` contains the agreed *Security Association* SA, a new random number N_i, optionally a new *Diffie-Hellman* value $[KE_i]$, and the traffic selectors TS_i and TS_r. As the calculation of new *Diffie-Hellman* values is a very resource-consuming task, we propose to omit this procedure for RFID tags. Nevertheless, if refreshing of the key is essential for security, the calculation of new DH values can be included in the protocol.

The reader sends an `IPsec_Child_SA` command to the RFID tag. As the payload of the `CREATE_CHILD_SA` request is encrypted and the header is authenticated, the whole packet has to be transmitted to the tag. The tag can decrypt the payload and verify the authenticity of the message. The `IPsec_Child_SA` response from the tag contains the accepted *Security Association* SA, a new random number N_r, and the traffic selectors TS_i and TS_r. The tag's response is translated into an `IPsec_Child_SA` response and sent back to the *Correspondent Node*. With the new random numbers and the previous derived master key SK_d, new keys are generated which are used in the following IPSec communication. Figure 6.8 shows phase two of the key-exchange protocol.

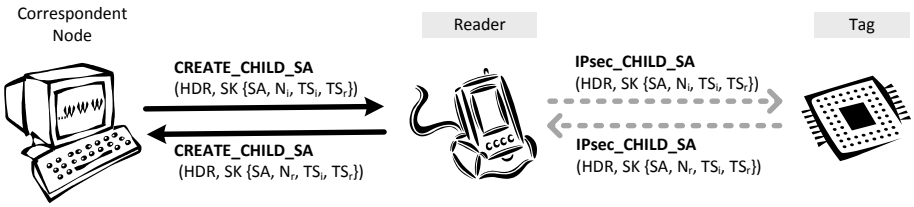


Figure 6.8: Create IPSec SA for RFID Tags

6.2.3 Secure End-to-End Communication

Once the IPSec keys are exchanged, secure communication can be establish. All IPv6 packets sent over the secure channel between the *Correspondent Node* and the tag are encrypted and authenticated using the exchanged keys. As we described in Section 6.2.1, we use *Encapsulating Security Payload* in *Transport Mode* as security policy for the communication. The structure of an IPv6 packet using the ESP header in *Transport Mode* is presented in Figure 6.2. Using this policy, confidentiality, message integrity, and authentication are provided.

Secure end-to-end communication works as follows: When the reader receives an protected IPv6 packet from the *Correspondent Node*, it generates an `ESP_Pkt` RFID command and sends it to the tag. The tag can decrypt and verify the payload and sends its response, which is again encrypted and authenticated, back to the reader. The reader builds a valid IPv6 packet inserting the tag's response as payload. The correct addresses are inserted and the packet is sent to the CN. As only the tag and the *Correspondent Node* know the session keys, end-to-end security is provided. The reader cannot neither modify the payload without detection nor read the content of the exchanged data. Figure 6.9 shows a secure IPSec session between tag and *Correspondent Node*. The IPv6

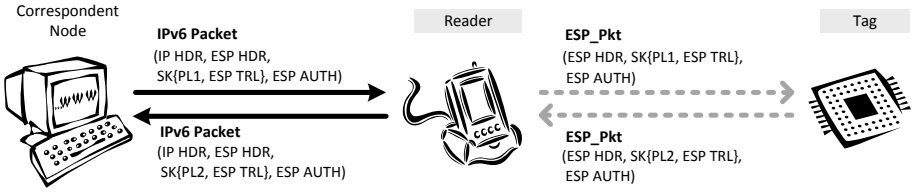


Figure 6.9: Secure End-to-End Communication with RFID Tag

packet received by the reader starts with the IP header, followed by the ESP header (`ESP HDR`). The next part of the packet is encrypted. This encrypted part holds the payload (`PL1`) , and the ESP trailer (`ESP TRL`). The last part of the packet is the ESP-authentication field (`ESP AUTH`), which is used to provide message integrity. The reader translates the IPv6 packet into an RFID command (`ESP_Pkt`), which contains the encrypted part of the packet as well as the ESP header and the ESP-authentication field.

The tag verifies the authenticity of the message and decrypts the encrypted part of the packet. It processes the payload `PL1` and generates a response to the received packet. The response (`PL2`) and a new ESP trailer are encrypted, and new ESP header and authentication fields are generated. These values are sent back to the reader, which appends an IP header with exchanged source and destination addresses in order to build an IPv6 response packet for the *Correspondent Node*.

Using the proposed concept, secure end-to-end communication with RFID tags on the Internet is provided. In the next section we consider the feasibility of implementing the required functionality in passive RFID tags. Furthermore, we discuss the security of the communication protocol in Section 6.2.5.

6.2.4 Feasibility for Passive RFID Tags

RFID tags have to provide advanced functionality to implement the proposed protocol. Various cryptographic primitives are available for the use in passive RFID tags. We have presented hardware implementations of common cryptographic algorithms in Section 2.1. In the following, we consider how secure online communication can be implemented on the tag by using these cryptographic primitives.

For secure end-to-end communication over an secure channel, which has been established before, symmetric encryption and authentication algorithms are used. We propose to use the AES algorithm for encryption and the the HMAC-SHA-1 algorithm for authentication. When considering the implementations described in Section 2.1.2, we are confident that both algorithms can be used in parallel on current passive RFID tags. Another possibility to provide encryption and authentication is to use the AES algorithm as stand-alone cryptographic primitive as specified in [Sch05]. Based on this RFC, the AES algorithm can be used for encryption, for authentication, and for random value generation.

The establishment of a secure channel is much more demanding than secure communication. The proposed key-exchange protocol is based on the *Diffie-Hellman* key-exchange algorithm [DH76]. In Section 2.1, we have presented a hardware implementation of a *Diffie-Hellman Authentication* [BBD+08]. Based on the findings of this publication, we consider a *Diffie-Hellman* key exchange and calculation of a *Diffie-Hellman* value feasible for passive RFID tags. The messages exchanged during the IKEv2 protocol are encrypted and authenticated. For providing these functions, AES and SHA-1-HMAC can be used, like for secure communication.

If certificates are used during the authentication phase of the key-exchange protocol, an asymmetric digital-signature algorithm has to be provided. Based on the type of public key, an appropriate algorithm has to be applied. We propose to use an ECDSA signature and verification algorithm for this purpose. Several hardware modules implementing this algorithm for passive RFID tags are available (c.f., Section 2.1).

Using the described cryptographic primitives, the proposed key exchange protocol as well as the secure communication protocol can be implemented on passive RFID tags. Although all of these algorithms are feasible for passive RFID tags when implemented as stand-alone algorithms, they could exceed the tag's capabilities when implemented in parallel on one tag. Also memory requirements for handling of the protocol have to be considered. In order to evaluate the implementation requirements of the proposed protocol, the development of a prototype providing all necessary functionality is in progress. Until this development is not finished, we cannot give reliable figures on the feasibility of the communication protocol for current RFID tags.

6.2.5 Security Considerations

The security of IPSec and the IKE protocol have been analyzed in many research articles (e.g., [Bel96, Mea99, PK00, GHT00, ARM07, TG05, Aur06]), which came to the conclusion that IPSec is at the moment the best IP security protocol available. As most of the security problems within IPSec are related to its complexity (outlined in [FS00]), we consider the proposed protocol, which is an even more restricted version of IPSec, more secure than the original one.

One main difference of the proposed protocol is the limitation to only one connection to the tag at a certain time. In this way we omit the *Security Associations* database. Furthermore, we allow only one protocol and mode of operation (ESP in *Transport Mode*), i.e., we omit all security concerns regarding the AH protocol and the *Tunnel Mode*. One issue that must be considered is the suggestion to use a timeout before establishing a new connection (see Section 6.2.1). An attacker could take advantage of this procedure to perform subsequent Denial-of-Service attacks. We can mitigate this threat by using randomized timeout values to prevent synchronization of the attackers activity, and by enabling the reader to block suspicious behavior of a *Correspondent Node*.

Identity management is another issue to consider for security. A problem that is often addressed in several works in this area, as, for instance, in [ARM07], is that IP addresses are used as both host identifiers and location addresses. Mismatches between the identifiers used in the *Security Policy Database* and the identifiers provided by the authentication protocol can occur. We can handle this problem by using IKEv2 with certificates. When using certificates, the identity of the sender can be verified, because a certificate links the identity to the public key of the sender. The infrastructure for a central *Certification Authority* is already established in the proposed concept: The *Home Agent* can issue certificates for the tags as well as for every *Correspondent Node*. The *Correspondent Node* is only able to establish a secure connection, if there is no mismatch in the identity used during key exchange and the identity given in the certificate. With the described assumptions we have made on the IPSec protocol, even a higher level of security can be provided than for the original one.

6.3 Conclusion

In Chapter 6, we discuss the establishment of a security layer based on the communication concept presented in Chapter 5. We base the proposed concept on the security protocol of the traditional Internet, namely the IPSec protocol. In order to adapt IPSec to the requirements of passive RFID tags, some assumptions have to made on the IP packet size, the number of connections, the traffic policy, and the use of certificates. In the proposed system, only one open connection at a time and only one traffic policy, namely *Encapsulating Security Payload* in *Transport Mode* are allowed. Certificates are handled by the *Home Agent* to unburden the tags from this task. We also limit the number of *Security Associations* to one per tag.

Applying the assumptions previously mentioned, the IKEv2 protocol as well as the IPSec protocol can be modified in order to enable key exchange and secure communication using IPv6 for passive RFID tags. As most of the security problems in IPSec originate from its complexity, the limitations we have applied to the protocol are supposed to have a positive effect on its security. The feasibility of an implementation of the protocol on current RFID tags has still to be investigated. Nevertheless, each cryptographic algorithm required to implement the protocol is available as hardware module feasible for passive RFID tags. The research on the feasibility of the proposed protocol on passive RFID tags is an open task. Implementation of a working tag prototype implementing secure communication on the Internet is ongoing. The outcome of this feasibility study will be an important step towards secure online RFID applications on the *Internet of Things*.

7 Conclusions and Outlook

In this doctoral thesis we discuss the basics for the development of secure applications involving passive RFID tags on the future *Internet of Things*. Attention goes to the three building parts essential to that development: cryptographic primitives for passive RFID tags, the design of secure applications using these primitives, and a concept on how to securely connect RFID tags to the Internet.

In order to identify cryptographic capabilities of current passive RFID tags limitations of power, time, and size are discussed. Taking these limitations into account, cryptographic hardware primitives that are feasible even for the constrained environment in such tags are presented. For providing security measures in RFID applications we propose to use standardized cryptographic algorithms and protocols, as they offer a high level of security and interoperability. With the presented hardware implementations of standardized encryption algorithms, hash algorithms and digital-signature algorithms, the basic security services confidentiality, integrity, authentication, and non-repudiation can be provided to RFID applications.

In order to find appropriate ways to secure an RFID application, threats to security have to be identified first. Due to the non-line-of-sight character of RFID and the automatic way tags interact with readers, RFID technology is very susceptible to remote attacks. We have found tracking and tracing, cloning and counterfeiting, and unauthorized tag access as the three main categories of security threats in RFID applications. Discussing appropriate countermeasures to each threat, we conclude that authentication is one of the most important security service that has to be provided for building secure RFID applications.

We present the design and evaluation of RFID applications that include authentication with standardized cryptographic algorithms and protocols. The development of the presented applications demonstrates the feasibility of au-

thentication mechanisms in real-world RFID systems and documents the practical impact of introducing security in RFID applications. A formal design flow for these applications with special attention to the integration of security mechanisms is presented. The design flow consists of six phases from the application specification to the production of the tags.

The design step of developing software models and simulation, as well as the prototyping step are essential parts in the design flow. Based on the findings in these steps, the outcome of former design phases, like application specification or security-enhanced communication protocol, can be modified and adapted to new requirements without much effort. Modifications in a later design phase would be much more costly. Therefore, successful simulation together with prototyping of the application proved to be a critical success factor of developing RFID applications.

In order to simplify and speed up the simulation and prototyping phases of the RFID-application design flow, simulation and prototyping tools can be employed. We present two of these tools, namely *PETRA* and *ProtEx*. The simulation tool *PETRA* is mainly used for developing software models of the tags and the reader application. The software models are used to verify the functionality and evaluate the performance of the application. The simulation and prototyping tool *ProtEx* is based on *PETRA*, as it uses parts of this tool as basis for simulation of RFID protocols. Nevertheless, the simulation capabilities of *PETRA* have been extended for *ProtEx*. Besides the simulation and development of software models, *ProtEx* is able to re-use the reader application and parts of the software model for prototyping. *ProtEx* was developed with special attention to a high level of reusability and user-friendliness. We integrated the IAIK *DemoTag*, which is a programmable RFID tag, into the design flow with *ProtEx* to create a first hardware prototype of the tag. With a physical reader and the programmable tag platform connected via the serial ports, the developer can use *ProtEx* to do a proof of concept in hardware very quickly.

ProtEx can be adapted to various RFID communication standards. The existing *ProtEx* variants support the ISO-18000-3 standard as well as the ISO-14443 standard. In the future, other communication standards will be implemented on *ProtEx* in order to extend its application range. Another task remains open to future research, namely to maximize the reusability between the tag software and the used tag prototype. At the moment modifications are inevitable due to different programming platforms. With these advances, prototyping of RFID applications will be even more efficient.

As online communication with passive RFID tags is an important feature for future RFID applications, we discuss the integration of these tags as mobile nodes into the *Internet of Things*. As we see the *Internet Protocol version 6* (IPv6) as the common language of the future Internet, a concept on connecting passive RFID tags using mobile IPv6 mechanisms is presented. The most important concepts from mobile IPv6 used in our proposal are described. In the presented communication concept, tags do not have to perform IPv6 processing

itself, but the reader works as router and translator for the tags in its field. The reader, which is computationally strong enough for this challenge, takes over the more complex tasks in IP communication. As RFID tags are often not present in any reader field, they are often offline. Therefore, a delayed-messaging approach has been proposed to address this issue. Another RFID-related issue is the tag-triggered communication, as in a standard RFID communication protocol the reader talks first. We present a method that helps to overcome this restriction. Using the proposed communication protocol, two-way communication with passive RFID tags on the Internet is enabled.

Various applications require secure interaction with RFID tags, especially if the tags are addressed remotely over the Internet. Secure communication on the conventional Internet is established by using *Internet Protocol Security* (IPSec). As RFID tags are limited devices, they cannot be expected to implement the standard IPSec protocol. Based on the proposed communication concept a modified IPSec protocol feasible for passive RFID tags is presented. The modified protocol involves the key exchange, which is based on the *Internet Key Exchange* protocol, as well as secure communication via IPv6. In this way, a secure end-to-end connection between a *Correspondent Node* and a passive RFID tag can be established.

For implementation of the key exchange and communication protocol, the tag has to support various cryptographic algorithms and protocols, namely the *Diffie-Hellman* key exchange, *Advanced Encryption Standard* (AES), and *Secure-Hash Algorithm* (SHA-1). Also the *Elliptic-Curve Digital-Signature Algorithm* (ECDSA) might be necessary if certificates are involved for authentication. All of these primitives are feasible for the use in passive RFID tags as stand-alone implementations. At the moment we have no data about the conjunctive use of all these algorithms on one tag. This fact leads us to the outlook on future activities in this area.

At the moment of writing this thesis, prototyping of the proposed communication concept is in progress. The results of this prototyping process will give feedback on the feasibility of the protocol implementation on current RFID tags. Even if the proposed protocol implementation exceeded current tag's capabilities, we are confident that future tags are able to provide the required functionality. This is due to the evolvement of extended tag capabilities based on new semi-conductor technologies and antenna development. Progression in tag capabilities and the research on even less power-consuming cryptographic primitives will lead to passive RFID tags communicating securely on the Internet. We claim that our contribution is an important advance on the way to an all-embracing *Internet of Things*.

Bibliography

[ADF07] Manfred Aigner, Sandra Dominikus, and Martin Feldhofer. A System of Secure Virtual Coupons Using NFC Technology. In *Workshop on Pervasive RFID/NFC Technology and Applications (PerTec07), New York, USA, March 19, 2007, Proceedings*, pages 362–366. IEEE, March 2007.

[ARM07] Tuomas Aura, Michael Roe, and Anish Mohammed. Experiences with host-to-host ipsec. In *Proceedings of the 13th international conference on Security protocols*, pages 3–22, Berlin, Heidelberg, 2007. Springer-Verlag.

[AT09] Gildas Avoine and Aslan Tchamkerten. An efficient distance bounding RFID authentication protocol: balancing false-acceptance rate and memory requirement. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *Information Security Conference – ISC'09*, volume 5735 of *Lecture Notes in Computer Science*, pages 250–261, Pisa, Italy, September 2009. Springer.

[Aue08] Andreas Auer. Scaling Hardware for Electronic Signatures to a Minimum. Master thesis, University of Technology Graz, October 2008.

[Aur06] Roe M. Aura, T. Designing the mobile ipv6 security protocol. *Annales des Telecommunications/Annals of Telecommunications*, 61(3-4):332–356, 2006. cited By (since 1996) 6.

[BBD+08] Holger Bock, Michael Braun, Markus Dichtl, Erwin Hess, Johann Heyszl, Walter Kargl, Helmut Koroschetz, Bernd Meyer, and Hermann Seuschek. A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography. Invited talk at RFIDsec 2008, July 2008.

[BCC04] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. In Pfitzmann Atluri and McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, volume 2004, pages 132–145, October 2004.

[Bel96] Steven M. Bellovin. Problem areas for the ip security protocols. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, pages 21–21, Berkeley, CA, USA, 1996. USENIX Association.

[BR07] Leonid Bolotnyy and Gabriel Robins. Physically Unclonable Function-Based Security and Privacy in RFID Systems. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2007), White Plains, New York, USA, 19-23 March, 2007, Proceedings*, pages 211–220. IEEE Computer Society, 2007.

[CDDJ] Jari-Pascal Curty, Michel Declercq, Catherine Dehollain, and Norbert Joehl.

[Com09] European Commission. Commission Recommendation on the Implementation of Privacy and Data Protection Principles in Applications Supported by Radio-Frequency Identification. 2009.

[CYCSM08] Lin Yu-Shan Chang Yao-Chung, Chen Jiann-Liang and Wang Shi-Ming. RFIPv6 - A Novel IPv6-EPC Bridge Mechanism. In *Proceedings of the IEEE International Conference on Consumer Electronics*, Januar 2008.

[DA07] Sandra Dominikus and Manfred Josef Aigner. mcoupons: An application for near field communication (nfc). In *AINA Workshops (2)*, volume AINA Workshops (2), pages 421–428. IEEE Computer Society, 2007.

[DAK10] S. Dominikus, M. Aigner, and S. Kraxberger. Passive rfid technology for the internet of things. In *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*, pages 1 –8, November 2010.

[Dem] http://www.iaik.tugraz.at/content/research/rfid/tag_emulators/.

[DGAK11] S. Dominikus, H. Gross, M. Aigner, and S. Kraxberger. Low-cost rfid tags as ipv6 nodes in the internet of things. In Ping Wang Tieyan Li, Chao-Hsien Chu and Guilin Wang, editors, *Radio Frequency Identification System Security, RFIDsec'11 Asia Workshop Proceedings*, volume Cryptology and Information Security Series, pages 114–128. IOSPress, April 2011.

[DH76] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DH98] S. Deering and A. Hiden. RFC 2460: Internet Protocol, Version 6 (IPv6) Specification, December 1998.

[DK11] Sandra Dominikus and Stefan Kraxberger. Secure communication with rfid tags in the internet of things. *Security and Communication Networks*, 2011.

[DM02] Sandra Dominikus and Stefan Mangard. Eine universelle AES Hardware Architektur. In Peter Söser and Andreas Buslehner, editors, *Proceedings of Austrochip 2002, October 4, 2002, Graz, Austria*, pages 65–72, October 2002. ISBN 3-9501635-0-6.

[DOF05] Sandra Dominikus, Elisabeth Oswald, and Martin Feldhofer. Symmetric authentication for RFID systems in practice. In *Workshop on RFID and Lightweigth Crypto, July 13-15, 2005, Graz, Austria*, 2005.

[DOF06] Sandra Dominikus, Elisabeth Oswald, and Martin Feldhofer. Practical Security for RFID: Strong Authentication Protocols. In Patrick Horster, editor, *Proceedings of D.A.C.H. Mobility 2006, October 17-18, 2006, Graz, Austria*, pages 187–200. Syssec, 2006. ISBN 3-00-019635-8.

[Dom02] Sandra Dominikus. A hardware implementation of MD4-family hash algorithms. In *9th IEEE International Conference on Electronics, Circuits and Systems, Dubrovnik, Croatia, 15-18 September, 2002, Proceedings*, volume 3, pages 1143–1146. IEEE, October 2002.

[Dom11] S. Dominikus. Medassist - a privacy preserving application using rfid tags. In *IEEE International Conference on RFID-Technology and Applications 2011*, September 2011.

[DSP+08] Srinivas Devadas, Edward Suh, Sid Paral, Richard Sowell, Tom Ziola, and Vivek Khandelwal. Design and Implementation of PUF-Based "Unclonable" RFID ICs for Anti-Counterfeiting and Security Applications. In *IEEE International Conference on RFID, Las Vegas, NV, USA, April 16-17, 2008, Proceedings*, pages 58–64. IEEE, April 2008.

[Eng02] Daniel W. Engels. Comparison of the Electronic Product Code Identification Scheme & the Internet Protocol Address Identification Scheme, June 2002.

[EPC03] EPCglobal. 13.56 MHz ISM Band Class 1 Radio Frequency (RF) Identification Tag Interface Specification, February 2003. Available online at `http://www.epcglobalinc.org/`.

[EPC07] EPCglobal. The EPCglobal Architecture Framework, September 2007.

[Eur05] European Commission - ARTICLE 29 Data Protection Working Party. Working document on data protection issues related to RFID technology, January 2005. Available online at `http://www.europa.eu.int/comm/privacy`.

[FAD05] Martin Feldhofer, Manfred Aigner, and Sandra Dominikus. An Application of RFID Tags using Secure Symmetric Authentication. In Panagiotis Georgiadis, Stefanos Gritzalis, and Giannis F. Marias, editors, *1st International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing – SecPerU 2005, Santorini Island, Greece, July 14, 2005, Proceedings*, pages 43–49. Diavlos Publications, July 2005. In conjunction with the IEEE ICPS'05.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong Authentication for RFID Systems using the AES Algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, August 2004.

[Fin03] Klaus Finkenzeller. *RFID-Handbook*. Carl Hanser Verlag, 2nd edition, April 2003. ISBN 0-470-84402-7.

[FM05] Elgar Fleisch and Friedemann Mattern. *Das Internet der Dinge: Ubiquitous Computing und RFID in der Praxis:Visionen, Technologien, Anwendungen, Handlungsanleitungen.* Springer-Verlag New York, Inc., 2005. ISBN 3540240039.

[FP08] Christian Floerkemeier and Ravikanth Pappu. Evaluation of RFIDSim - a Physical and Logical Layer RFID Simulation Engine. In *Proceedings of IEEE International Conference on RFID 2008*, 2008.

[FR06] Martin Feldhofer and Christian Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *First International OTM Workshop on Information Security (IS'06), Montpellier, France, Oct 30 - Nov 1, 2006. Proceedings, Part I*, volume 4277 of *Lecture Notes in Computer Science*, pages 372–381. Springer, October 2006.

[FS00] N. Ferguson and B. Schneier. A Cryptographic Evaluation of IPsec. Technical report, 3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA, 2000.

[FSL04] Christian Floerkemeier, Roland Schneider, and Marc Langheinrich. Scanning with a Purpose – Supporting the Fair Information Principles in RFID Protocols. In Hitomi Murakami, Hideyuki Nakashima, Hideyuki Tokuda, and Michiaki Yasumura, editors, *International Symposium on Ubiquitous Computing Systems – UCS 2004*, volume 3598 of *Lecture Notes in Computer Science*, pages 214–231, Tokyo, Japan, November 2004. Springer.

[FW07] Martin Feldhofer and Johannes Wolkerstorfer. Strong Crypto for RFID Tags - Comparison of Low-Power Hardware Implementations. In *IEEE International Symposium on Circuits and Systems (ISCAS 2007), New Orleans, USA, May 27-30, 2007, Proceedings*, pages 1839–1842. IEEE, May 2007.

[FWR05] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings on Information Security*, 152(1):13–20, October 2005.

[FWS08] Christian Floerkemeier, Matthias Wille, and Sanjay Sarma. RFIDSim - A Physical and Logical Layer Simulation Engine for Passive RFID. *IEEE Transactions on Automation - Special Issue RFID*, 2008.

[GHT00] Joshua D. Guttman, Amy L. Herzog, and F. Javier Thayer. Authentication and confidentiality via ipsec. In *Proceedings of the 6th European Symposium on Research in Computer Security*, pages 255–272, London, UK, 2000. Springer-Verlag.

[GJP05] Simson Garfinkel, Ari Juels, and Ravi Pappu. RFID Privacy: An Overview of Problems and Proposed Solutions. *IEEE Security and Privacy Magazine*, 3(3):34–43, May-June 2005.

[GJR07] Marc Girault, Loc Juniot, and Matthew Robshaw. The feasibility of On-the-Tag Public Key Cryptography. In Jorge Munilla, Alberto Peinado, and Vincent Rijmen, editors, *Workshop on RFID Security 2007 (RFIDSec07), July 11-13, Malaga, Spain*, pages 77–86, July 2007.

[Han10] Gerhard P. Hancke. Design of a Secure Distance-Bounding Channel for RFID. *Journal of Network and Computer Applications*, May 2010.

[HC98] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.

[HFP10] Michael Hutter, Martin Feldhofer, and Thomas Plos. An ECDSA Processor for RFID Authentication. In Berna Ors, editor, *Workshop on RFID Security – RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9, 2010, Proceedings*, Lecture Notes in Computer Science. Springer, 2010.

[HK05] Gerhard Hancke and Markus Kuhn. An RFID Distance Bounding Protocol. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005), Athens, Greece, 5-9 September 2005, Proceedings*, pages 67–73. IEEE Computer Society, September 2005.

[HWF08] Daniel Hein, Johannes Wolkerstorfer, and Norbert Felber. ECC is Ready for RFID - A Proof in Silicon. In *Workshop on RFID Security 2008 (RFIDsec08)*, July 2008.

[Int93] International Organisation for Standardization (ISO). Information Technology - Security Techniques - Entity authentication mechanisms - Part 3: Entity authentication using a public key algorithm, 1993.

[Int99] International Organisation for Standardization (ISO). ISO/IEC 9798-2: Information technology – Security techniques – Entity authentication – Mechanisms using symmetric encipherment algorithms, 1999.

[Int00] International Organization for Standardization (ISO). ISO/IEC 14443: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards, 2000.

[Int01] International Organisation for Standardization (ISO). ISO/IEC 15693-3: Identification cards - Contactless integrated circuit(s) cards - Vicinity cards – Part 3: Anticollision and transmission protocol, 2001.

[Int04a] International Organisation for Standardization (ISO). ISO/IEC 18092: Information technology - Telecommunications and information exchange between systems - Near Field Communication - Interface and Protocol, April 2004.

[Int04b] International Organization for Standardization (ISO). ISO/IEC 18000-3: Information Technology AIDC Techniques — RFID for Item Management – Part 3: Parameters for air interface communications at 13.56 MHz, March 2004.

[Int04c] International Organization for Standardization (ISO). ISO/IEC 18000-6: Information Technology AIDC Techniques — RFID for Item Management – Part 6: Parameters for air interface communications at 860-960 MHz, 2004.

[JPA04] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775 (Draft Standard), June 2004.

[JRS03] Ari Juels, Ronald L. Rivest, and Michael Szydlo. The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy. In *10th ACM Conference on Computer and Communication Security, Washington, DC, USA, October 27-30, 2003, Proceedings*, pages 103–111. ACM Press, October 2003.

[JSB05] Ari Juels, Paul Syverson, and Dan Bailey. High-Power Proxies for Enhancing RFID Privacy and Utility. In George Danezis and David Martin, editors, *Workshop on Privacy Enhancing Technologies (PET 2005), Cavtat, Croatia, May 30-June 1, 2005, Proceedings*, volume 3856 of *Lecture Notes in Computer Science*, pages 210–226. Springer, 2005.

[Jue04] Ari Juels. Minimalist Cryptography for Low-Cost RFID Tags. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, volume 3352 of *Lecture Notes in Computer Science*, pages 149–164. Springer, September 2004.

[Jue06] Ari Juels. RFID Security and Privacy: A Research Survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, February 2006.

[JW06] Ari Juels and Stephen A. Weis. Defining Strong Privacy for RFID. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2006/137, April 2006.

[Kau05] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), December 2005. Obsoleted by RFC 5996, updated by RFC 5282.

[Ken05a] S. Kent. IP Authentication Header. RFC 4302 (Proposed Standard), December 2005.

[Ken05b] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), December 2005.

[KF10] Thomas Kern and Martin Feldhofer. Low-Resource ECDSA Implementation for Passive RFID Tags. In *17th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2010), December 12-15th, 2010, Athens, Greece, Proceedings*, pages 1236–1239. IEEE, 2010.

[KHNE10] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), September 2010. Updated by RFC 5998.

[KKBD11] Süleyman Kardaş, Mehmet Sabir Kiraz, Muhammed Ali Bingöl, and Hüseyin Demirci. A Novel RFID Distance Bounding Protocol Based on Physically Unclonable Functions. In *Workshop on RFID Security – RFIDSec'11*, Amherst, Massachusetts, USA, June 2011.

[KRCJ06] Mooseop Kim, Jaecheol Ryou, Yongje Choi, and Sungik Jun. Low Power AES Hardware Architecture for Radio Frequency Identification . In Hiroshi Yoshiura, Kouichi Sakurai, Kai Rannenberg, Yuko Murayama, and Shinichi Kawamura, editors, *First International Workshop on Security (IWSEC 2006), Kyoto, Japan, October 23-24, 2006, Proceedings*, volume 4266 of *Lecture Notes in Computer Science*, pages 353–363. Springer, October 2006.

[KS05] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040.

[Lan09a] Marc Langheinrich. A survey of rfid privacy approaches. *Personal and Ubiquitous Computing (Special Issue)*, 13(6):413–421, August 2009.

[Lan09b] Marc Langheinrich. *Ubiquitous Computing*, chapter Privacy in Ubiquitous Computing, pages ??–?? CRC Press, 2009.

[MAD03] Stefan Mangard, Manfred Aigner, and Sandra Dominikus. A Highly Regular and Scalable AES Hardware Architecture. *IEEE Transactions on Computers*, 52(4):483–491, April 2003.

[Mea99] C. Meadows. Analysis of the internet key exchange protocol using the nrl protocol analyzer. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 216–231, 1999.

[MOP06] Jorge Munilla, Andres Ortiz, and Alberto Peinado. Distance Bounding Protocols with void-challenges for RFID. In *Workshop on RFID Security 2006 (RFIDSec06), July 12-14, Graz, Austria*, 2006.

[MSW05] David Molnar, Andrea Soppera, and David Wagner. A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, Ontario, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 276–290, Kingston, Canada, August 2005. Springer.

[MW04] David Molnar and David Wagner. Privacy and Security in Library RFID: Issues, Practices, and Architectures. In *11th ACM Conference on Computer and Communications Security (CCS), Washington, DC, USA, October, 2004, Proceedings*, pages 210–219. ACM Press, October 2004.

[Nat00] National Institute of Standards and Technology (NIST). FIPS-186-2: Digital Signature Standard (DSS), January 2000. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat01] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat02] National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.

[NNSS07] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007.

[O'N08] Màire O'Neill. Low-Cost SHA-1 Hash Function Architecture for RFID Tags. In Sandra Dominikus, editor, *Workshop on RFID Security 2008 (RFIDsec08)*, pages 41–51, July 2008.

[OSK03] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic Approach to "Privacy-Friendly" Tags. RFID Privacy Workshop, November 2003. Available online at `http://lasecwww.epfl.ch/~gavoine/download/papers/OhkuboSK-2003-mit-paper.pdf`.

[Pap01] Ravikanth S. Pappu. *Physical one-way functions*. PhD thesis, Massachusetts Institute of Technology, March 2001.

[PK00] Radia Perlman and Charlie Kaufman. Key exchange in ipsec: Analysis of ike. *IEEE Internet Computing*, 4:50–56, November 2000.

[PLHCETR07] Pedro Peris-Lopez, Julio C. Hernandez-Castro, Juan M. Estevez-Tapiador, and Arturo Ribagorda. LAMED - A PRNG for EPC Class-1 Generation-2 RFID Specification. *Computer Standard & Interfaces, Elsevier*, February 2007.

[Plo07] Thomas Plos. Implementation of a Security-Enhanced Semi-Passive UHF RFID Tag. Master's thesis, Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria, May 2007.

[PMDW05] Norbert Pramstaller, Stefan Mangard, Sandra Dominikus, and Johannes Wolkerstorfer. Efficient AES Implementations on ASICs and FPGAs. In H. Dobbertin, Vincent Rijmen, and A. Sowa, editors, *Fourth Workshop on the Advanced Encryption Standard, AES4 - State of the Crypto Analysis', Bonn, Germany, May 10-12, 2005, Proceedings*, volume 3373 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2005.

[RCT05] Melanie Rieback, Bruno Crispo, and Andrew Tanenbaum. RFID Guardian: A Battery-Powered Mobile Device for RFID Privacy Management. In Colin Boyd, González Nieto, and Juan Manuel, editors, *Australasian Conference on Information Security and Privacy – ACISP'05*, volume 3574 of *Lecture Notes in Computer Science*, pages 184–194, Brisbane, Australia, July 2005. Springer.

[REC04] Damith C. Ranasinghe, Daniel W. Engels, and Peter H. Cole. Security and Privacy: Modest Proposals for Low-Cost RFID Systems. In *Auto-ID Labs Research Workshop, Zurich, Switzerland, September 2004, Proceedings*, September 2004.

[RNTS06] Jason Reid, Juan Gonzalez Neito, Tee Tang, and Bouchra Senadji. Detecting Relay Attacks with Timing Based Protocols. Cryptology QUT ePrint Archive, (`http://eprints.qut.edu.au/`), Report 3264, February 2006.

[SBA00] Sanjay Sarma, David L. Brock, and Kevin Ashton. White Paper: The Networked Physical World. MIT-AUTOID-WH-001.pdf, October 2000.

[Sch05] J. Schiller. Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2). RFC 4307 (Proposed Standard), December 2005.

[SDMKHJ07] Lee Sang-Do, Shin Myung-Ki, and Kim Hyoung-Jun. EPC vs. IPv6 mapping mechanism. In *The 9th International Conference on Advanced Communication Technology*, volume 2, pages 1243–1245, February 2007.

[SE03] Sanjay E. Sarma and Daniel W. Engels. Technical Report: On the Future of RFID Tags and Protocols, June 2003.

[SE09] Sarah Spiekermann and Sergei Evdokimov. Privacy Enhancing Technologies for RFID - A Critical Investigation of State of the Art Research. In *IEEE Privacy and Security*. IEEE, IEEE Computer Society, 2009.

[SVW10] Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. PUF-Enhanced RFID Security and Privacy. In *Secure*

Component and System Identification – SECSI'10, Cologne, Germany, April 2010.

[TG05] Jonathan Trostle and Bill Gossman. Techniques for improving the security and manageability of ipsec policy. *International Journal of Information Security*, 4:209–226, 2005. 10.1007/s10207-004-0064-6.

[TNJ07] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), September 2007.

[VB03] István Vajda and Levente Buttyán. Lightweight Authentication Protocols for Low-Cost RFID Tags. In *2nd Workshop on Security in Ubiquitous Computing, in conjunction with Ubicomp 2003, Seattle, Washington, USA, October 12, 2003, Proceedings*, pages 1–10, October 2003.

[Wei91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.

[Wol05] Johannes Wolkerstorfer. Scaling ECC Hardware to a Minimum. In Nikolaus Kerö and Peter Rössler, editors, *Proceedings of Austrochip 2005, October 6, 2005, Vienna, Austria*, pages 207–214, October 2005. ISBN 3-901578-13-7.

[WSRE03] Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, and Daniel W. Engels. Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In Dieter Hutter, Günter Müller, Werner Stephan, and Markus Ullmann, editors, *Security in Pervasive Computing, 1st Annual Conference on Security in Pervasive Computing, Boppard, Germany, March 12-14, 2003, Revised Papers*, volume 2802 of *Lecture Notes in Computer Science*, pages 201–212. Springer, March 2003.

[WYY05] Xiaoyun Wang, Yiqun Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Victor Shoup, editor, *Advances in Cryptology a CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin / Heidelberg, 2005. $10.1007/11535218_2$.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

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 marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)