FAKULTÄT FÜR !NFORMATIK

Securing Backbones for Embedded Home and Building Automation Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

ausgeführt von

Daniel Lechner
Matrikelnummer 0325612

am:
Institut für Rechnergestützte Automation

Betreuung:
Betreuer: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Dipl. Ing. Mag. Wolfgang Granzer

Wien, 22. Juli 2009

(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract

Home and Building Automation (HBA) systems control the traditional building services lighting and shading as well as heating, ventilation and air conditioning (HVAC). The main benefits of using HBA systems are the efficient use of resources with the following energy savings and the increased comfort and wellness.

Up to now, other building service types like access control, alarm and fire alarm systems have only been realised by dedicated stand-alone systems. Today a tighter integration of these isolated systems is desirable, since data which is available in one system can also be used by other ones. This integration increases the security needs of HBA systems, which have been underrated in common protocols up to now. At the same time, the use of IP backbone networks to interconnect field networks becomes more common. Unfortunately, these IP based networks are even more attractive for adversaries and prone to security attacks.

This thesis deals with the protection of communication on HBA backbone networks. The principle goal is to support embedded devices, since mainly devices with a small amount of memory and computational power are used in HBA installations. First, cryptographic principles and concepts are explained. Afterwards, state of the art protocols are discussed and rated with respect to the special requirements of HBA backbones. The main part of this thesis forms the elaboration of a new concept, satisfying these special security requirements. The proposed solution performs the necessary key management as well as the protection of the communication. While the protocol was designed with HBA networks in mind, it is also applicable to other application domains (e.g., industrial automation).

Zusammenfassung

Unter dem Betriff Heim- und Gebäudeautomation (HGA) versteht man die Steuerung und Regelung unterschiedlicher im Gebäude befindlicher Geräte, wie Lichtanlagen, Beschattung sowie Heizungs-, Lüftungs und Klimageräte (HLK). Der Vorteil von HGA ist einerseits eine effizientere Nutzung der vorhandenen Ressourcen und die daraus resultierende Energieeinsparung, als auch eine deutliche Komfortsteigerung und Steigerung des Wohlbefindens.

Bis jetzt waren andere Systeme wie Zutrittskontrolle, Alarm- oder Brandmeldeanlagen komplett von der HGA getrennt. Eine stärkere Integration ist aber wünschenswert, da Daten eines Systems auch innerhalb eines anderen verwendet werden können. Durch diese Verschmelzung der Systeme steigen auch die Sicherheitsanforderungen, die in den gängigen HGA Standards bis jetzt großteils vernachlässigt wurden. Gleichzeitig werden vermehrt IP Netzwerke für das Verbinden mehrerer HGA Subnetze verwendet, was wiederum eine größere Angriffsfläche für potentielle Angreifer bietet.

Aus diesem Grund beschäftigt sich diese Diplomarbeit mit der Absicherung der Daten auf dem Backbone eines HGA Systems. Einen Schwerpunkt bildet dabei die Unterstützung von ,,embedded Devices'', da hauptsächlich Geräte mit wenig Rechenleistung und Speicher in HGA Installationen verwendet werden. Zu Beginn werden kryptographische Grundlagen und Konzepte vorgestellt. Nach einer eingehenden Untersuchung bereits etablierter Sicherheitsmechanismen, werden diese in Bezug auf die speziellen Anforderungen eines HGA Systems bewertet. Der Hauptteil dieser Arbeit beschäftigt sich mit der Ausarbeitung eines neuen Konzepts, das sowohl das geforderte Maß an Sicherheit bietet, als auch mit den besonderen Anforderungen des Anwendungsbereiches zurechtkommt. Die vorgestellte Lösung behandelt sowohl die erforderliche Verwaltung der Schlüssel, als auch die abgesicherte Kommunikation im Betrieb des Systems. Das Protokoll und die Implementierung sind so konzipiert, sodass diese nicht nur in HGA Systemen, sondern auch in anderen Anwendungsbereichen angewendet werden können.

Danksagung

Ich möchte mich an dieser Stelle bei allen herzlich bedanken, die mich in der Zeit meines Studiums, sowie bei dieser Arbeit tatkräftig unterstützt haben. Ohne die Hilfe all meiner Wegbegleiter hätte diese Arbeit nicht entstehen können.

Im Speziellen gilt dies für meine Freundin EDITH, die öfter meine Launen und Stimmungen ertragen musste und es immer wieder schaffte mich aufzumuntern. Weiters gilt mein Dank meinen Eltern EDITH und ROBERT, sowie meinem Bruder OLIVER, die mir meine Ausbildung ermöglicht und mich bei jeder Gelegenheit unterstützt haben.

Besonderer Dank gilt auch meinen Betreuern WOLFGANG GRANZER und WOLFGANG KASTNER, die mich in jeder Phase des Projekts mit Ideen und Hilfestellungen unterstützt und mir und meiner Arbeit sehr viel Zeit gewidmet haben. Die ausgezeichnete Athmosphäre am Institut hat mir sehr geholfen, mit Spaß und Engagement bei der Sache zu bleiben. In diesem Sinne möchte ich auch HARALD WEILLECHNER, FRITZ PRAUS, CHRISTIAN REINISCH und GEORG NEUGSCHWANDTNER danken, die mir immer mit ihrer fachlichen Kompetenz zur Seite standen, mich auf neue Wege brachten und gemeinsam mit meinen Betreuern entscheidend zur guten Stimmung beigetragen haben.

Für die nötige Abwechslung möchte ich mich bei meinen Vereinskollegen von ,,Dreh und Trink Ultimate Frisbee'' bedanken. Der Sport und das Vereinsleben sind ein wichtiger Bestandteil meiner Freizeit geworden und haben oft geholfen, den Kopf frei zu bekommen.

This thesis has been carried out within the project *Security in Building Automation* (FWF Österreichischer Fonds zur Förderung der wissenschaftlichen Forschung; Projekt P19673).

Contents

Abstract i

Zusammenfassung ii

Danksagung iii

1 Introduction 1

1.1 Motivation . . . 1
1.2 KNX and KNXnet/IP . . . 3
1.3 Requirements . . . 4
1.4 Contribution . . . 5

2 Cryptography 7

2.1 History . . . 7
2.2 Terminology . . . 8
2.3 Entity Authentication . . . 9
2.4 Confidentiality . . . 10
2.4.1 Symmetric encryption . . . 10
2.4.2 Asymmetric encryption . . . 14
2.5 Integrity . . . 24
2.5.1 Hash functions . . . 25
2.5.2 Digital signature . . . 27
2.6 Freshness . . . 34
2.7 Elliptic Curve Integrated Encryption Scheme (ECIES) . . . 35
2.8 Randomness . . . 36
2.9 Key management . . . 40
2.9.1 Pre-shared . . . 40
2.9.2 Symmetric . . . 41

2.9.3 Asymmetric . . . 42
2.10 Securing multicast . . . 47

3 Security for Internet Protocol (IP) backbones in Home and Building Automation (HBA) 49
3.1 Internet Protocol Security (IPsec) . . . 49
3.1.1 Security Association (SA) . . . 50
3.1.2 Authentication Header (AH) . . . 50
3.1.3 Encapsulated Security Payload (ESP) . . . 51
3.1.4 Modes of operation . . . 51
3.1.5 Internet Key Exchange (IKE) . . . 51
3.1.6 IPsec and multicast . . . 52
3.1.7 Conclusion . . . 53
3.2 SSL/TLS . . . 53
3.2.1 Cipher suites . . . 54
3.2.2 Secure Sockets Layer (TLS) handshake . . . 54
3.2.3 Conclusion . . . 57
3.3 VPN . . . 57
3.4 Secure Shell (SSH) . . . 58
3.5 Summary . . . 60

4 Solution 61
4.1 Configuration using Engineering Tool Software (ETS) . . . 62
4.2 Key set distribution . . . 63
4.2.1 Unicast . . . 63
4.2.2 Multicast . . . 64
4.2.3 Blacklist . . . 69
4.2.4 Certificate revocation . . . 70
4.3 Secure communication . . . 71
4.3.1 Key set revocation . . . 72
4.4 Security analysis . . . 72
4.4.1 Replay attacks . . . 74

5 Implementation 76
5.1 Prototype implementation . . . 76
5.1.1 Test configuration . . . 76

5.1.2 Router architecture 77
5.1.3 Security token representation 83
5.1.4 Message representation 84
5.2 Benchmark 88

6 Conclusion and Outlook **90**

A Acronyms **92**

B Bibliography **95**

C Code snippets **101**
C.1 Keyed-Hash Message Authentication Code (HMAC) 101
C.2 Diffie-Hellman (DH) 102
C.3 Frame building 103
C.4 `uIP` patches 105
C.5 Other examples 107

List of Figures

1.1 HBA network 2

2.1 Secure transformation 9
2.2 Secure transformation with a key 9
2.3 Encryption and decryption 10
2.4 Symmetric encryption and decryption 11
2.5 Exemplary use of Initialisation Vector (IV) (CBC mode) 12
2.6 Stream cipher 13
2.7 Asymmetric encryption and decryption 15
2.8 Integrity protection 24
2.9 Unkeyed hash function 25
2.10 Keyed hash function 27
2.11 Digital signature 28
2.12 Data freshness 35
2.13 ECIES encryption 36
2.14 ECIES decryption 37
2.15 Linear Feedback Shift Register (LFSR) example 38
2.16 Symmetric key management 42
2.17 Certificate creation 43
2.18 Certificate verification 43
2.19 DH principle 44
2.20 DH man-in-the-middle 45
2.21 Group Diffie-Hellman (GDH) network requirements 46
2.22 Multicast 48

3.1 Position of IPsec in the Open Systems Interconnection (OSI) layer model 50
3.2 Position of TLS in the OSI layer model 54
3.3 full TLS handshake 55

3.4 TLS Client-/Server Hello message 55
3.5 OpenVPN multicast 58
3.6 Position of the SSH protocol in the OSI layer model 59

4.1 Protocol position in the OSI layer model 61
4.2 Configuration phase 62
4.3 Key set distribution for unicast services 64
4.4 Coordinator scenarios 65
4.5 Conflicting coordinator scenarios 68
4.6 Inform blacklisted device 70
4.7 Revocation scenarios 72

5.1 Prototype network 77
5.2 Hardware and software overview 78
5.3 Key distribution main state machine 81
5.4 Routing indication frame format 82
5.5 Public key representation 83
5.6 Digital signature representation 84
5.7 Certificate representation 84
5.8 Key set representation 84
5.9 Message representation 85
5.10 Key distribution message representation 86
5.11 Secured frame format 88

List of Tables

2.2 LFSR example 39
2.3 Comparison of group key distribution protocols 47

3.2 Comparison of security protocols 60

5.1 Available service types 85
5.3 Key distribution message types 86
5.5 Benchmark results 89

Listings

C.1 HMAC code 101
C.2 DH code 102
C.3 Frame building and symmetric encryption 103
C.4 `uIP` multicast patch 105
C.5 `uIP` multicast patch (header) 106
C.6 `uIP` multicast patch (Adress Resolution Protocol (ARP)) 106
C.7 Flash memory write example 107

Chapter 1

Introduction

1.1 Motivation

The core application area of Home and Building Automation (HBA) systems is environmental control with the traditional building services lighting and shading as well as heating, ventilation and air conditioning (HVAC). Up to now, other building service types such as safety-critical (e.g., fire alarm systems) and security-critical services (e.g., access control, intrusion alarm systems) have only been realised by dedicated stand-alone systems.

Today, a tighter integration of these formerly separated systems is desirable. However, to integrate safety- and security-critical applications into HBA installations, the underlying control network has to be protected against malicious interference. An important step towards a secure HBA system is to secure the communication i.e., all exchanged data within the HBA network.

HBA networks are typically implemented following a two-tier model. Field networks are home for sensors, actuators, and controllers that interact with the environment and perform measurement and control tasks. These field networks are interconnected by a common backbone where management nodes (e.g., operator workstation, logging server) that require a global view of the entire HBA network are located. Figure 1.1 shows a typical example of such a topology.

At the field level, robustness and flexibility is most important. Therefore, transport media like Twisted Pair (TP) or Power Line (PL) cabling, or wireless technologies that communicate via Radio frequency (RF) are used.

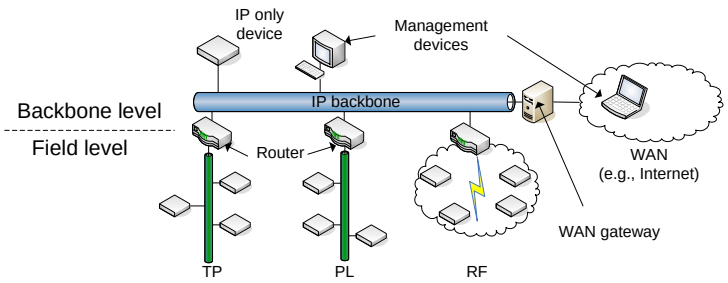


Figure 1.1: HBA network

At the backbone level, it is more common to use high performance network media. Today, a trend towards the use of IP based networks as backbone can be observed. The main reason is that due to widespread use of IP based networks (especially Ethernet based networks) in the IT world, the costs for cabling and hardware for network interface controllers are rapidly decreasing. Therefore, it becomes economically feasible to extend embedded microcontrollers with a dedicated Ethernet interface chip. Another reason for the popularity of IP based backbones is that many buildings already provide an existing IP infrastructure (e.g., office Local Area Network (LAN)) which can be shared with the HBA network.

With the integration of safety- and security-critical applications, the communication within the HBA networks has to be protected against security attacks. Especially the IP backbone is prone to security attacks. Because of its widespread use, the protocol is well-known and many security flaws have been discovered. Unfortunately, the IP protocol itself as well as the underlying network medium (e.g., Ethernet) do not natively provide any security mechanisms. Nevertheless, many well-established security mechanisms have been built on top of the IP protocol. Since these mechanisms have been developed primarily for the IT domain, they cannot directly be used in HBA systems. The main difference between the IT and the HBA world is the use of embedded systems, since small size and low power consumption is very important in the HBA domain. More and more devices of everyday's life are equipped with embedded devices, touching almost every area of life, ranging from cell phones, cars to kitchen appliances and many more. In contrast to the devices used in the IT domain, embedded devices offer limited processing power and memory capacity. These limiting factors and special requirements of the HBA networks are the main reasons, why available security mechanisms cannot be used to protect HBA systems.

Research activities at the Automation Systems Group which is part of the Institute of Computer Aided Automation at the Technical University of Vienna show, that security has been neglected in HBA systems. An open standard that completely lacks support for security is KNX (cf. [1]). Therefore the aim of this thesis is to find mechanisms to secure IP backbones of KNX installations.

1.2 KNX and KNXnet/IP

KNX is a popular standard for HBA systems, especially in Europe (cf. [2]). Besides TP, PL and RF as communication medium, also IP networks can be used to interconnect several subnets. This protocol extension called KNXnet/IP (cf. [3]) provides different mechanisms to encapsulate KNX network messages into IP messages.

The KNXnet/IP specification [3] is divided into 8 parts, where the parts 2 – 8 specify different services:

Core services. Besides the definition of basic KNXnet/IP, this part includes the specification of services for KNXnet/IP device discovery and connection management.

Device management. This part specifies services like read and write of KNXnet/IP specific properties for the remote management of KNXnet/IP servers .

Tunneling. KNXnet/IP clients can use the tunneling service to establish a management connection to a KNXnet/IP server.

Routing. KNXnet/IP routers use the KNXnet/IP routing service to forward messages between local KNX network segments over the IP network.

Remote logging. This part defines services for offline-monitoring of KNX networks.

Remote configuration and diagnosis. Here, configuration and diagnosis services are defined.

Object Server. In this part, services to access a KNXnet/IP object server are defined. A KNXnet/IP client can use these services to access the communication objects on the KNXnet/IP object server.

In addition to these encapsulation mechanisms, there are investigations underway that use IP directly as a native KNX medium [4] on the field level.

1.3 Requirements

To fully protect the communication within a KNXnet/IP network, different security mechanisms have to be integrated into the KNXnet/IP protocol. Regarding the demands on a secure KNXnet/IP network, the following objectives can be identified:

Mutual entity authentication. To prevent an impersonation of legitimate KNXnet/IP devices, the involved KNXnet/IP servers and clients must prove their identity before they can securely communicate with each other.

Secured channel. To protect the transmitted data that is exchanged between the authenticated communication partners, a secured channel has to be provided. Such a channel uses physical and/or cryptographic techniques to guarantee different security objectives. Depending on the security requirements of the application, these are data integrity, freshness and/or confidentiality (cf. Section 2.2).

Support for unicast and multicast. Section 1.2 lists a number of different communication services of KNXnet/IP. The main goal of a security extension useable for KNXnet/IP is to fully support all of these KNXnet/IP services. While most of them are based on unicast communication, some services like KNXnet/IP routing and the KNXnet/IP discovery service are based on multicast communication. Therefore, a security extension has to be used that provides support for unicast and multicast communication.

Low processing power and memory consumption. Providing security by physical techniques (e.g., immuring the network cables) is not always easy to achieve. In such a case, cryptographic techniques have to be used. For reasons of cost efficiency, embedded devices with limited processing power and limited memory are commonly used in KNX installations. Since cryptographic techniques are computationally intensive, their use must not exceed the available device resources.

Currently, KNXnet/IP does not specify security mechanisms that fulfill the stated requirements for secure communication. In Chapter 1 of the KNXnet/IP specification, various security threats are briefly discussed and some rudimentary guidelines are presented. Most of them are based on an isolation of the KNXnet/IP network (e.g., firewall, KNXnet/IP only Intranet) and on ''Security by Obscurity'' (e.g., use non-standard multicast address, rely on the missing expertise of an attacker). However, since pre-

venting physical access to the network by isolation is not always possible (e.g., in office LANs, WLANs) and "Security by Obscurity" is a technique that only temporarily provides protection (if at all), these suggested organisational measures cannot fully satisfy the security requirements for security systems.

Therefore, a new concept is necessary to secure KNXnet/IP systems, which will be presented in this thesis.

1.4 Contribution

Due to the absence of security mechanisms in KNXnet/IP, this thesis presents a protocol for the secure key exchange process and secure communication. It is suitable for constrained devices and fulfills special network requirements typically found in HBA systems. In contrast to other works like [5], which focuses on application security, this thesis is focused on communication security. In [6] and [7], a security extension called EIBsec for KNX TP 1 networks has been presented. With little adaption, EIBsec can also be applied to other KNX network media. However, it is tailored to the characteristics of KNX TP 1 (e.g., limited length of standard KNX frames) and dedicated to secure KNX frames. Therefore, the KNXnet/IP specific part of KNXnet/IP frames (e.g., KNXnet/IP header) as well as services that are only available for KNXnet/IP (e.g., KNXnet/IP device discovery) cannot be secured using EIBsec.

This thesis gives a basic introduction to the field of cryptography in Chapter 2. Hash functions and the two major fields symmetric and asymmetric en- and decryption (together with their digital signature pendants) and their mathematical background are explained. Some representatives of each category are introduced – especially those used in the solution. Additionally examples, as well as pros and cons of each method are mentioned. Furthermore, this chapter deals with the closely related topics key management, randomness and the different ways to secure multicast communication.

Afterwards, already available and well established state-of-the-art protocols like IPsec and TLS are examined with respect to the special requirements of the HBA domain.

Since it turned out, that none of these available protocols satisfy all requirements, Chapter 4 brings up a possible solution. Besides the standard use case, also conflicting and probably problematic scenarios and configurations are mentioned, which can be

handled by the protocol. A detailed security analysis of the new protocol closes Chapter 4.

The last major part of this thesis contains the prototype implementation. Chapter 5 describes the hardware and software architecture as well as implementation details. A benchmark at the end of this chapter presents the usability of the software implementation in real HBA networks. Although this prototype implementation uses KNX and KNXnet/IP as underlying HBA standard, ongoing research activities have shown, that the presented security concept is not limited to KNX. In [8], it is shown that the developed security extension is also applicable to other HBA standards like LonWorks (cf. [9]) and BACnet (cf. [10]). Furthermore, the use of this extension in other domains like other control networks and networks with embedded devices is possible.

Chapter 2

Cryptography

2.1 History

Secure communication has been a topic for thousands of years. While in early days primarily only head of states and warlords tried to hide their messages from others, nowadays almost every user of the Internet is used to utilise cryptography – even if most of them might not think about it.

It is known that already in the 5^{th} century B.C. cryptographic algorithms have been used to secure messages. The characters of a message were mixed up to build an anagram and became unreadable to eavesdropper. This method is called transposition. An alternative to the transposition is the substitution or shift cipher. Julius Caesar, who was very engaged on the field of cryptography, was one of the first who used substitution, which is also called Caesar's cipher. Basically this kind of algorithm has been used for a long time. When a cryptanalyst found a weak point of an encryption, cryptographers enhanced it by one or more different stages and adaptions – like the use of one-time keys.

Even the Enigma machine, mostly used by German forces during the World War II, used an advanced form of substitution. The main innovation of the Enigma machine was the mechanically automated en- and decryption of messages. It formed the most secure communication system in these days. But also cryptanalysts began to use automated systems. The Polish mathematician Marian Rejewski was able to break the strong encryption scheme of the Enigma machine and built a machine to automatically derive the used encryption keys. The decryption machine developed by the Polish

scientists has been improved and adopted by Alan Turing according to changes in the Enigma machine during the war. According to statements of different politicians, the work of the cryptanalysts form the basis of the outcome of the World War II.

The development of modern computers in the 20th century significantly affected the field of cryptography. Cryptographers can use them to execute difficult algorithms very comfortable and in reasonable time. But also cryptanalysts can start better attacks against these algorithms.

Therefore almost all today's cryptographic algorithms are designed with respect to Kerckhoff's assumption (cf. [11]). It has to be assumed, that the cryptanalyst has complete knowledge of the cryptographic algorithm and implementation. This means, that the secrecy must reside entirely in the key.

2.2 Terminology

In the basic scenario, a sender wants to send a message to a receiver. The message is referred to as plaintext – sometimes also called cleartext. Depending on the used transport channel, someone may be able to read or even alter the sent message. So the sender and the receiver will take some countermeasures to form a secure communication.

Defined by the application, there are different demands on secure communication. The author of [12] lists typical security objectives, which form the basis for the following compilation:

Entity Authentication. An attacker should not be able to masquerade as someone else. The receiver(s) must be able to verify that the message was really sent by the declared sender. On the other hand, the sender must be able to verify, that the message is only received by authorised receiver(s). This method of both-sided entity authentication is called mutual entity authentication (cf. [11]).

Confidentiality. Information should only be accessible to those parties, which are authorised to have access. In our basic scenario, only the authenticated receiver(s) with adequate access rights can read the message, but no one else.

Integrity. The receiver(s) of a message should be able to verify that the message has not been altered in transit.

Freshness. An adversary should not be able to replay an old message. The receiver(s) will only accept messages, which have been created recently by the sender. Retransmitted messages will be identified and discarded.

To provide these security objectives, different algorithms have to be used. To ensure one or more of the mentioned security objectives, a secure transformation has to be applied to the plaintext. To recover the plaintext from the secured text, a inverse transformation has to be carried out (cf. Figure 2.1).

Figure 2.1: Secure transformation

As already mentioned in Section 2.1, relying only on the nondisclosure of the algorithm is not considered to be secure for a long time. Modern cryptography solves this with the usage of (one or more) secret keys (cf. Figure 2.2).

Figure 2.2: Secure transformation with a key

In the following sections, different types of secure transformations will be introduced in detail. Each of them ensures one ore more of the listed security objectives.

2.3 Entity Authentication

[13] describes the most common technique to achieve entity authentication as verification of a message (which is possibly the response to a previous one), which demonstrates that the sender is in possession of some secret information. If the verification is successful (i.e. the secret is correct), the receiver of the message can conclude, that the sender is the one he pretends to be. For example, the secret information can contain

some password or Personal Identification Number (PIN), data stored on smart cards or human physical characteristics.

Another possibility of entity authentication is via a Trusted Third Party (TTP). Two entities authenticate themselves via a third party, i.e. they ask the third party, if the other party is the one it pretends to be.

Very often, entity authentication is done implicit. If two parties know the same key, which can be used e.g. for confidentiality (cf. Section 2.4) or integrity protection (cf. Section 2.5), these two parties trust each other, since only authenticated entities know this key. For this reason, entity authentication and key management (cf. Section 2.9) are closely related topics.

2.4 Confidentiality

To ensure confidentiality, the secure transformation has to completely hide the contents of a message from a third party. The whole message has to be altered (i.e. encrypted – cf. Figure 2.3) by the sender applying the secure transformation and the (authorised) receivers have to be able to perform an inverse transformation (decryption) to reconstruct the original message.

Figure 2.3: Encryption and decryption

2.4.1 Symmetric encryption

In symmetric algorithms, formally it is easy to derive the encryption key when the decryption key is known and vice versa. Usually the same key is used for encryption and decryption (cf. Figure 2.4). The main advantage is the high speed of these algorithms during operation due to the short keys. The main problem within symmetric algorithms is the key management (calculation, distribution and storage of keys – cf. Section 2.9). The channel for exchanging the keys has to provide entity authentication,

confidentiality, integrity and freshness (cf. Figure 2.4). The key(s) have to be present at all participating parties. This means, that it is sufficient for an attacker to compromise one end. If an attacker knows the secret key, he can encrypt the sent messages and create new decrypted messages without being noticed.

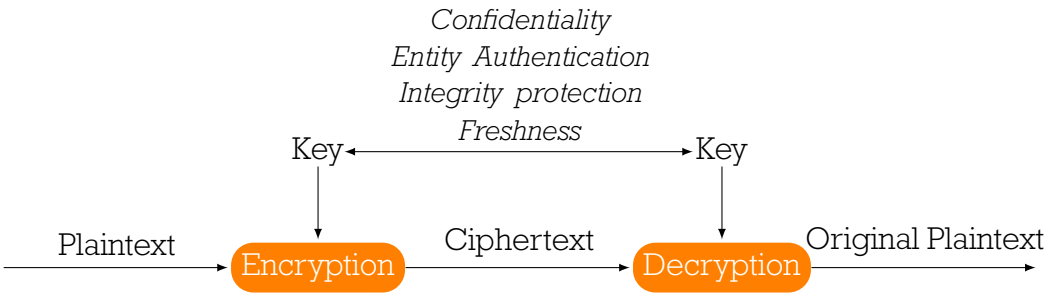


Figure 2.4: Symmetric encryption and decryption

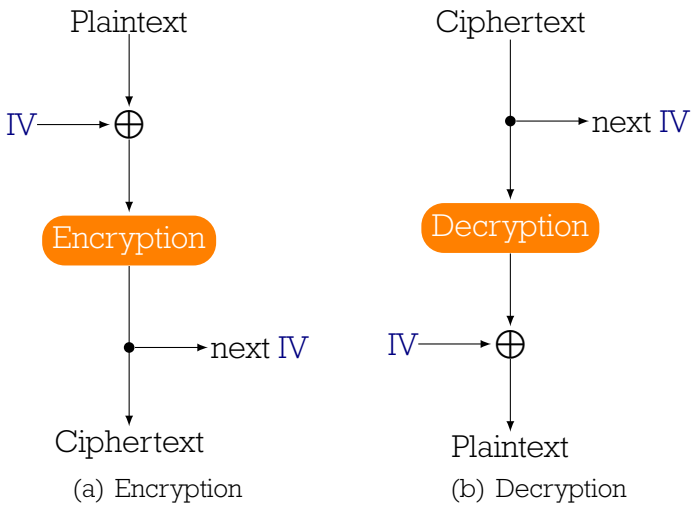
Since the ciphertext only depends on the plaintext message, the encryption key and the algorithm itself, same plaintext messages will bring up same ciphertext messages, if the key remains unchanged (which is usually the case for a certain time). As a consequence, replay of old messages could be possible and cryptanalysts may gain some useful information.

To overcome this problem, some data is used to linearly modify the plaintext or the ciphertext message. This linear modification is typically a bit by bit Exclusive OR $\oplus$ (XOR) operation (cf. Figure 2.5). The data, which is used for the modification, is usually randomly generated for the first block and called Initialisation Vector (IV). For the following blocks, data of the previous ones are used to generate the next IVs (cf. [11]).

Symmetric algorithms can be split up into two groups. The block ciphers encrypt only blocks of a fixed size. Stream ciphers encrypt the digits of the plaintext instantly and one by one.

Block cipher

Block ciphers encrypt only blocks of a fixed size and the resulting ciphertext has the same length. Longer plaintext messages are usually fragmented. Depending on the operation mode of a block cipher, the last block may be padded up to the necessary block length. For block ciphers, many different operation modes have been invented.



(a) Encryption (b) Decryption

Figure 2.5: Exemplary use of IV (CBC mode)

These modes mainly specify the handling of larger messages, how keys are applied and the application and generation of the IVs. Figure 2.5 shows the usage of the IV in Cipher Block Chaining (CBC) mode. The ciphertext of the previous data block is used as IV for the next one.

The National Institute of Standards and Technology (NIST) has currently approved eight modes of operation for symmetric block ciphers (cf. [14]): Electronic Codeblock (ECB), CBC, Output Feedback (OFB), Cipher Feedback (CFB), Galois/Counter Mode (GCM), Cipher-based MAC (CMAC), Counter with CBC-MAC (CCM) and Counter (CTR). The proper mode must be chosen based upon the security requirements and the length of plaintext data. E.g. CBC offers better performance for bigger data blocks, while CFB runs faster with smaller ones.

Advanced Encryption Standard (AES) - Rijndael The AES is a symmetric algorithm, which has been announced by the American NIST in [15]. The selected algorithm was the winner of an official competition. While the originally as Rijndael published algorithm can handle messages of different block sizes, the NIST fixed the block size to 128 bit. The specification knows three different versions of AES; AES-128, AES-192 and AES-256 with key lengths of 128, 192 and 256 bits.

AES interprets the plaintext block as two dimensional table. This table has 4 rows and 4 columns, with one byte of the plaintext in each cell. The encryption is done in several rounds. Some of the rounds expand and modify the used encryption key, while others

perform monoalphabetic substitutions, row shifts or mixes up the columns. Depending on the key size, these operations are performed several times on the plaintext table.

During decryption, all operations are performed in the reversed order to reconstruct the plaintext message.

Stream cipher

Stream ciphers convert the plaintext message instantly to ciphertext. Since every bit is processed immediately, these ciphers are best suited for data streams with unknown or missing data structure.

Figure 2.6 shows a simple example of a stream cipher. The encryption key is used as input for a keystream generator. The keystream generator produces pseudo-random digits (cf. Section 2.8), which are combined with the plaintext stream. In the presented example, the keystream generator only depends on the input key and its internal state. Other variants may also use the plaintext or ciphertext streams as additional inputs. The quality of the stream cipher heavily depends on the quality of the keystream generator.

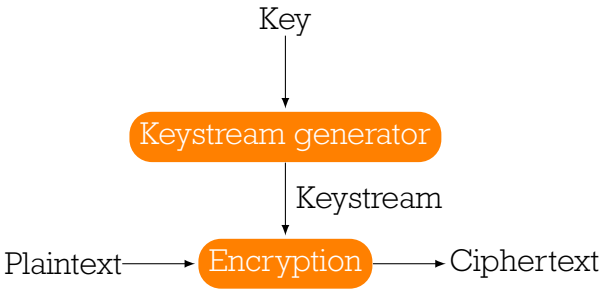


Figure 2.6: Stream cipher

RC4 RC4 is a stream cipher which has been developed by Ron Rivest in 1987. A description of the algorithm can be found in [11]. The used key can be of variable size. The cipher is built up like shown in Figure 2.6. The encryption operation is a bit wise XOR operation. As usual for stream ciphers, the complexity is embedded in the keystream generator.

The keystream generator only uses the key to calculate the keystream. It has a 8x8 S-box containing the elements $S_0, S_1, \ldots, S_{255}$. Initially, these elements are filled with

the sequence 0 to 255. At startup, depending on the used key, the S-box elements are mixed up. During operation, depending on the internal state, two elements are selected and the sum of these elements represent the output byte. Afterwards the used elements are swapped.

One of the biggest problems of the presented RC4 algorithm is, that the reinitialisation of the algorithm results in the same keystream (when using the same key). As an example, let K denote the keystream(s) used for the encryption. M_1 and M_2 are the plaintext messages to be encrypted. The ciphertext messages C_1 and C_2 are calculated via

$$\begin{aligned} C_1 &= M_1 \oplus K \\ C_2 &= M_2 \oplus K \end{aligned}$$

The XOR operation on the ciphertext messages result in the XOR operation on the plaintext messages.

$$C_1 \oplus C_2 = M_1 \oplus M_2$$

This result can be used to recover the plaintext messages by letter frequency analysis or other basic techniques (cf. [16]). If the algorithm is enhanced by an IV, this attack is not possible any more.

RC4 is used in many applications like WEP, WPA and for PDF encryption. Nevertheless, RC4 is not recommended for the use in new applications. Even though there is a huge number of possible states (about 2^{1700}), some attacks have been issued to break the algorithm. For example, [17] presents several weaknesses in the keystream generation of RC4.

2.4.2 Asymmetric encryption

The basic concepts of asymmetric algorithms, which are also known as public-key algorithms, were invented by Whitfield Diffie and Martin Hellman in 1976 and were published in [18].

Public key algorithms use different keys for encryption and decryption (cf. Figure 2.7), where only one of these keys has to be private. The other key (public key) is usually publicly available. Compared to symmetric algorithms, asymmetric ones are slow, but

can offer easier secure key-exchange mechanisms (cf. Section 2.9.3). The channel to exchange the public keys does not have to provide confidentiality.

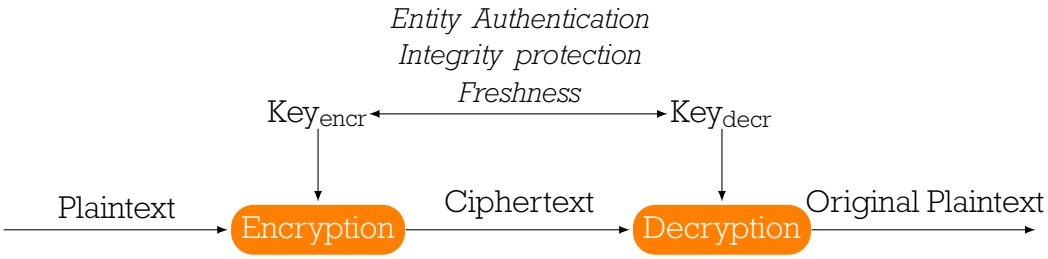


Figure 2.7: Asymmetric encryption and decryption

An one-way function f calculates the result $y = f(x)$ to a given input x with relatively less effort. But it is hard to reverse the function and calculate the input x, when the function f and the result y are known. Basically, trapdoor one-way functions have the same properties. But in addition it is easy to calculate the input x, when the function f, the result y and a secret are known. These trapdoor one-way functions are used in asymmetric cryptographic algorithms.

For encryption and decryption using asymmetric algorithms, the public key is used for the encryption of the message and the private key has to be used for the decryption. If Bob wants to send a message to Alice, he has to get to know Alice's public key in advance. With the help of this public key, he can encrypt the message and send it to Alice. In order to decrypt the message, Alice has to use the decryption algorithm together with her private key to read the contents of the message. A third person Eve, who does not know the private key (but maybe the public key), is not able to decrypt Bob's message to Alice.

Asymmetric cryptographic algorithms can be categorised by the underlying mathematical problem. This work will present algorithms based on the Integer Factorisation Problem (IFP), the Discrete Logarithm Problem (DLP) and the Elliptic Curve Discrete Logarithm Problem (ECDLP).

Integer Factorisation Problem (IFP)

One class of asymmetric algorithms are based on the difficulty of factoring large numbers.

Every positive integer greater than one can be written as product of prime numbers. This factorisation is always unambiguous. There are some factoring algorithms, but it is still very time consuming to find the prime factors of a number (cf. [11]). While a factorisation of 42 ($= 2 \cdot 3 \cdot 7$) is not very difficult (e.g. by trying every prime until $\sqrt{42}$), the topic becomes infeasible for big numbers. For cryptographic applications, primes with hundreds of digits are used.

RSA RSA was one of the first asymmetric cryptographic algorithm and probably the most important one. It has been developed in 1977 and is published in [19]. RSA can be used for encryption of data and for digital signatures (cf. Section 2.5.2).

Keypair generation During the RSA key generation, a private and a public key are generated. The public key consists of a pair of integers (n, e), where n is the RSA modulus and e the encryption exponent. n is a product of two randomly generated prime numbers p and q (of same bit length; $p \neq q$ and p and q should not be too close).

$$n = p \cdot q \tag{2.1}$$

These numbers (p and q) remain secret to the generator. The exponent e is a random integer, satisfying

$$\phi(n) = (p-1) \cdot (q-1) \tag{2.2}$$

$$1 < e < \phi(n) \tag{2.3}$$

$$gcd(e, \phi) = 1 \tag{2.4}$$

The function $\phi(n)$ (cf. Equation 2.2) is known as Euler's totient function. Assuming (for some a) that $lcm(a,n) = 1$, the totient function is used in Euler's theorem:

$$a^{\phi(n)} \equiv 1 \mod n \tag{2.5}$$

The private key d is the integer satisfying

$$1 < d < \phi(n) \tag{2.6}$$

$$e \cdot d \equiv 1 \mod \phi(n) \tag{2.7}$$

The values of p, q and $\phi(n)$ are not needed anymore and should be removed from the memory.

A small example will demonstrate the key generation. Alice wants to generate a RSA key pair and chooses two prime numbers

$$\begin{aligned} p &= 17 \\ q &= 11 \end{aligned}$$

The value of n is determined through Equation 2.1:

$$n = 17 \cdot 11 = 187$$

Euler's totient function results according to Equation 2.2 to

$$\begin{aligned} \phi(n) &= (p-1) \cdot (q-1) \\ &= 16 \cdot 10 \\ &= 160 \end{aligned}$$

Alice can choose the exponent

$$e = 7$$

Therefore, Alice's public key is $(187, 7)$. The private key d is calculated via Equation 2.7:

$$\begin{aligned} 7 \cdot d &\equiv 1 \mod 160 \\ d &= 23 \end{aligned}$$

To calculate the private key, Alice has to find the multiplicative inverse of $7 \mod 160$ (which is 23) e.g. by using the Extended Euclidean Algorithm.

Encryption/Decryption For the encryption of a message, only the public key of the receiver has to be used. A plaintext message M has to be transformed by an agreed function (e.g. the ASCII code) to the number m (with $m < n$). The ciphertext c is calcu-

lated by

$$c \equiv m^e \mod n \tag{2.8}$$

For the decryption of the message, the receiver's private key is used to derive the plaintext message:

$$m \equiv c^d \mod n \tag{2.9}$$

Simple conversion of the formulas show, that Equation 2.9 reconstructs the original value of m:

$$c^d \equiv (m^e)^d \mod n = m^{ed} \mod n \quad (\rightarrow \text{Eqn. 2.8})$$

From Equation 2.7 and with the help of Euler's theorem (assuming that $lcm(a,n) = 1$) follows:

$$\begin{aligned} m^{ed} \mod n &= m^{1+k\phi(n)} \mod n \quad (\rightarrow \text{Eqn. 2.7}) \\ &= m(m^k)^{\phi(n)} \mod n \\ &= m \mod n \quad (\rightarrow \text{Eqn. 2.5}) \\ m \mod n &= m \text{ (since } m < n) \end{aligned}$$

Back in the example, Bob wants to send a message to Alice. He uses the public key of Alice and calculates the ciphertext c with the help of Equation 2.8. In this example, the message $m = 14$.

$$\begin{aligned} c &\equiv 14^7 \mod 187 \\ &= 108 \end{aligned}$$

After transferring the ciphertext to Alice, she can decrypt the message using her private key via the Equation 2.9:

$$\begin{aligned} m &\equiv 108^{23} \mod 187 \\ &= 14 \end{aligned}$$

The attacker Eve who intercepts the message c is not able to derive m with reasonable

effort because she does not know Alice's private key.

Security analysis The most efficient mean to break the RSA security is the calculation of the prime factors p and q, given the public key (n, e) (and probably a ciphertext c). The integer factorisation of n provides these prime factors (cf. Equation 2.1) and enables an attacker to calculate the private key d through the Equations 2.2 and 2.7. Up to now, an efficient algorithm for solving this integer factorisation is not publicly known – it is still infeasible if the numbers are large enough and the algorithms based on this fact remain ''secure''.

Thus, the basic concept of RSA is still thought to be secure. Many vulnerabilities published up to now are based on an improper use of RSA and bad implementations.

Although, the presented basic RSA algorithm is not semantically secure. I.e. an attacker has the possibility to gather some information about the plaintext message m out of the given triple (N, e, c) (cf. [20, 21]).

Proper precomputations and preprocessings of the plaintext, which are necessary to secure RSA encryption, will not be discussed here since only a brief overview of available cryptographic algorithms should be given.

Discrete Logarithm Problem (DLP)

Another mathematical problem which is used for cryptographic algorithms is the DLP. Given a prime number p and the integers g and x, it is relatively easy to calculate the value of M via the equation $g^x \equiv M \mod p$. This operation is called discrete exponentiation. The inverse operation – the discrete logarithm – is rather complex. It is difficult to find the smallest possible value of x, when M, g and p are given.

DLP is the basis for many cryptographic algorithms like the Diffie-Hellman (DH) key agreement and ElGamal.

ElGamal ElGamal was published in 1985 in [22]. Like RSA, ElGamal uses computations on natural numbers (in prime fields). It can be used for the en- and decryption of messages as well as for digital signatures (cf. Section 2.5.2).

The ElGamal algorithm forms the basis for other encryption schemes, like the ECIES, which will be described in Section 2.7.

Keypair generation The calculations of ElGamal are based on a finite prime group with a dedicated prime number p. One entity has to choose a prime number p and two random numbers g and x (with $g < p$ and $x < p$). While x denotes the private key, the public key consists of the triple (y, g, p). The missing value y can be calculated via the following equation:

$$y \equiv g^x \mod p \tag{2.10}$$

The values of g and p can be shared among other parties in a group to generate their own private and public keys.

As a small example, Alice chooses the following numbers

$$\begin{aligned} p &= 23 \\ g &= 12 \\ x &= 7 \end{aligned}$$

The value of y is calculated with the Equation 2.10

$$y \equiv 12^7 \mod 23 = 16$$

The private key of Alice is 7 and the public key is the triple $(16, 12, 23)$.

Encryption/Decryption To encrypt a message m (with $m < p$), the sender has to choose another random number k, which satisfies

$$k < p \tag{2.11}$$

$$gcd(k, p) = 1 \tag{2.12}$$

The ciphertext can be calculated via

$$c_1 \equiv g^k \mod p \tag{2.13}$$

$$c_2 \equiv y^k m \mod p \tag{2.14}$$

where the triple (y, g, p) is the public key of the receiver. The tuple (c_1, c_2) represents the ciphertext. The size of the ciphertext is twice the size of the plaintext.

To decrypt the ciphertext, the receiver has to use its private key x. Following equation

$$m \equiv \frac{c_2}{{c_1}^x} \mod p \tag{2.15}$$

reconstructs the plaintext message m. The chosen random value k is not used for the decryption. It is important, that k is kept secret and is used only once. If a third person Eve knows a used k, she is able to derive Alice's private key x. Even if Eve gets two encrypted messages, where the same value for k was used, she is able to recover x.

With the help of the Equations above, the validity of Equation 2.15 can be shown:

$$\begin{aligned}
\frac{c_2}{{c_1}^x} \mod p &= \frac{y^k m}{g^{kx}} \mod p \quad (\rightarrow \text{Eqn. 2.13 and Eqn. 2.14}) \\
&= \frac{g^{kx} m}{g^{kx}} \mod p \quad (\rightarrow \text{Eqn. 2.15}) \\
&= m \mod p \\
m \mod p &= m \ (\text{since } m < p)
\end{aligned}$$

Continuing the previous example, Bob knows the public key of Alice and wants to send the message $m = 14$ to her. Bob chooses the random number

$$k = 21$$

and calculates the ciphertext with help of the Equations 2.13 and 2.14.

$$\begin{aligned}
c_1 &\equiv 12^{21} \mod 23 \equiv 2 \\
c_2 &\equiv 16^{21} \cdot 83 \mod 23 \equiv 21
\end{aligned}$$

When Bob sends the ciphertext $(2, 21)$ to Alice, she can decrypt it using her private key and Equation 2.15

$$\begin{aligned}
m &\equiv \frac{21}{2^7} \mod 23 = \frac{21}{13} \mod 23 = 21 \cdot 13^{-1} \mod 23 \\
&= 21 \cdot 16 \mod 23 \equiv 14
\end{aligned}$$

Like during the keypair generation of RSA (cf. Section 2.4.2), to calculate the plaintext,

Alice has to find the multiplicative inverse of 13 (which is 16) e.g. by the Extended Euclidean Algorithm.

Security analysis Basically, the mentioned security weaknesses of RSA also apply to ElGamal: the concept is still known to be secure, while the presented solution offers working surface for attackers (e.g. break semantic security). Especially when ElGamal is used to encrypt small secret keys which are used for symmetric encryption later on, it is often possible to derive the secret key from the ciphertext (cf. [21]). Therefore, this simple solution will not be used in real applications and serves for demonstration purposes only.

Elliptic Curve Discrete Logarithm Problem (ECDLP)

Elliptic Curve Cryptography (ECC) brought asymmetric encryption to low power embedded devices. While other asymmetric cryptographic methods based on IFP or DLP need a vast number of time and power consuming computations, ECC provides the same level of security with much cheaper calculations. Compared to the other presented algorithms, ECC algorithms are based on the algebraic structure of elliptic curves over finite fields (cf. [23]).

An elliptic curve is usually defined over the prime finite field F_p containing p elements or over the characteristic 2 finite field containing 2^m elements. The use of finite binary fields suggests itself, when implementing the algorithms on computer systems. Despite the lack of support for binary polynomial field arithmetic in embedded CPUs (cf. [24]), the implementations are more efficient than the ones over prime fields (cf. Section 5.2).

The finite field F_p is specified by an odd prime p. The elliptic curve $E(F_p)$ defined by the two parameters $a, b \in F_p$ (satisfying $4a^3 + 27b^2 \not\equiv 0 \mod p$) consists of all points $P(x,y)$, where $x, y \in F_p$ are solutions to the equation

$$y^2 \equiv x^3 + ax + b \mod p \tag{2.16}$$

The basic operation for elliptic curve cryptography is the scalar multiplication. Given a point $P(x,y) \in E(F_p)$ and an integer k, the scalar multiplication $k \cdot P$ is the addition of P to itself k times. The security in ECC is based on the fact that the equation $Q = kP$, with

$Q, P \in F_p$ and $k \in \mathbb{N}$, is easy to calculate if k and P are given. But it is infeasible to derive k, given Q and P. This is called the ECDLP. Since the basic security primitive of ECC is similar to the previously presented ones, in principle it is easy to port algorithms towards ECC. E.g. there are analogues of the RSA and the ElGamal algorithms [13]. An encryption scheme which uses ECC primitives can be found in Section 2.7.

A point on the curve can be stored in two different ways. The obvious representation is to describe a point with it's coordinates x and y. But if x is known, one can find y by solving the curve equation (Equation 2.16). Since this equation is a quadratic one, there are two possible solutions for y. So storing an extra bit in addition to x completely defines the location of the point. This representation is called ''point compression'', since it saves almost half of the space needed for point storage and transmission.

The main benefit of ECC is the use of much smaller keys, resulting in a lower memory usage and reduced calculation overhead. Based on today's knowledge, a 160 bit ECC key provides a better level of security than a 1024 bit RSA key (cf. [25]).

Domain parameters The domain parameter $D = (q, \mathrm{FR}, S, a, b, P, n, h)$ describe the used elliptic curve E and consists of (cf. [23]):

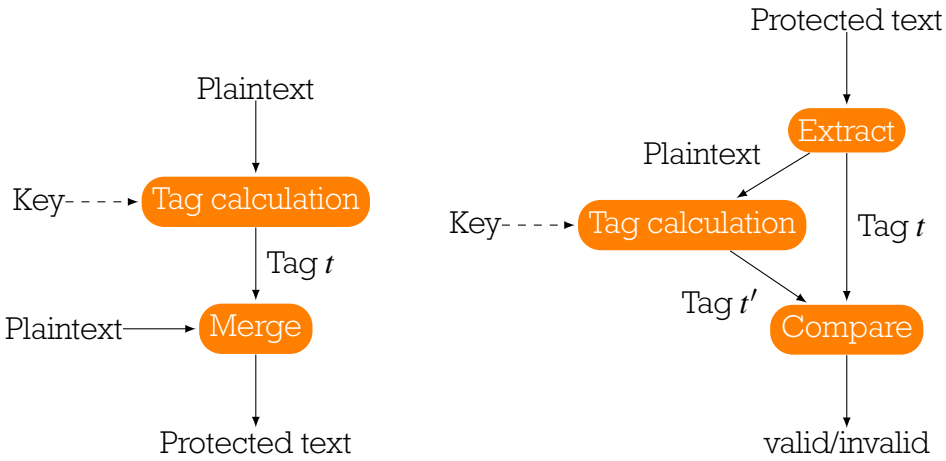
1. The *field order* q.
2. The *field representation* (FR) indication.
3. The *seed* S, if the curve was randomly generated.
4. Two *coefficients* $a, b \in \mathbb{F}_q$ (cf. Equation 2.16).
5. The *base point* P, which is described by the two *field elements* x_P and y_P; $P = (x_P, y_P) \in E(\mathbb{F}_q)$.
6. The *order* n of the base point P.
7. The *cofactor* $h = \#E(\mathbb{F}_q)$.

The choice of correct domain parameters is very crucial, since the security of the used encryption algorithms depend on these parameters. By the use of insecure domain parameters the system may be vulnerable to several attacks. These parameters can for instance lead to a radical reduction of the effort for reversing an underlying one-way function.

Elliptic curves, which are known to be secure (up to now), can be found in many publications like [26, 27] and are recommended by the Standards for Efficient Cryptography Group (SECG) and the NIST.

2.5 Integrity

To protect the integrity of a message, a small amount of data called tag[1] has to be transmitted in addition to the message itself. This tag identifies the message and an alternation of the original message can be recognised.



(a) Tag creation (b) Tag verification

Figure 2.8: Integrity protection

Figure 2.8 shows one possible basic setup of integrity protection. If Alice wants to send a message to Bob and protect it against alternation, she has to calculate a tag t of the message. To transmit the tag to Bob, in this example Alice merges (e.g. appends) the tag with the original message (cf. Figure 2.8(a)). When Bob receives the whole message, he can extract the tag t and use the same tag calculation as Alice to compute a reference tag t' out of the received plaintext. If $t = t'$, the tag is valid (cf. Figure 2.8(b)). Note that this simple kind of tag transmission only protects against transmission or storage errors. If an attacker is able to modify the message and the tag (and knows the tag calculation function), the modification will not be recognised by Bob (cf. Section 2.5.1).

[1] Depending on the used algorithm, other terms like checksum, message digest, hash value, ... are used in literature.

2.5.1 Hash functions

Hash functions are one-way functions. As described in Section 2.4.2, with an one-way function f it is possible to calculate the result $y = f(x)$ to a given input x with relatively less effort. But it is hard to reverse the function and calculate the previous input x, when the function f and the result y are known.

Hash functions convert an input string into a fixed length output string – called hash value – which is generally smaller than the input string. Therefore hash values are usually ambiguous.

The functions are surjective, but not injective: If two messages have different hash values, these messages are different. If two messages have the same hash values, it is more or less likely (depending on the used hash function and the size of the hash value) that these two messages are identical. Since the hash value is usually smaller than the original message, it has a much smaller value domain. Therefore it is not possible to determine certainly, if two messages are exactly the same (cf. [11]). If it is computationally infeasible to find a message with a certain hash value, the algorithm is called collision-free.

Hash values are used during the generation of digital signatures of messages (cf. Section 2.5.2). To generate the signatures, only the hash value of a message is processed instead of the whole message. Because of the independence of the bits within a hash value, hash functions can also be ''abused'' to generate random numbers or to generate (temporary) key sets.

Generally, hash functions do not involve any types of keys to generate the hash value (cf. Figure 2.9). The output only depends on the input string and the function itself. These unkeyed hash functions can be used to provide integrity only in a restricted manner. Only in environments where either the message or the hash value can be modified, integrity can be established by unkeyed hash functions. If an attacker knows the used hash function and he is able to alter the message and the hash value, he can modify messages without being noticed.

Figure 2.9: Unkeyed hash function

Secure Hash Algorithm (SHA)

SHA is the algorithm, which implements the Secure Hash Standard (SHS). It has been designed by the NIST and was published in 1993 and revised in 1995 (SHA-0 and SHA-1). These two versions of the algorithm are not recommended any more (however they are still used in many existing applications). While for SHA-0 collisions have been found and were published e.g. in [28], theoretical attacks against SHA-1 can be found in publications like [29]. Nevertheless, for compatibility reasons, SHA-1 is still optionally used in many protocols like TLS and IPsec (cf. Chapter 3).

Their successors form the SHA-2 family (cf. [30]), which are state of the art today. While using an identical algorithm, the length of the hash value is variable and part of the synonyms SHA-224, SHA-256, SHA-386 and SHA-512. Up to now, none of the SHA-2 algorithms have been broken.

The algorithm operates on blocks with fixed size (512 or 1024 bits). Within several rounds (64 or 80), the message blocks are processed with the internal state by logical operations to calculate the message hash.

The next version SHA-3 is already in development. Similar to the selection process of AES, the NIST has announced a competition for SHA-3. The winner will be nominated in 2012.

Section 2.5.2 shows one application field of SHA hash functions.

Keyed-Hash Message Authentication Code (HMAC)

Keyed hash functions are one-way hash functions with the addition of a secret key (cf. Figure 2.10). The hash value depends on the input string as well as on the used secret key. Again, the hash value can be used to ensure integrity. Now, the attacker has to know the key to generate a correct hash value to a modified message. With keyed hash functions, one can also ensure entity authentication: only the authenticated sender, who knows the key, are able to generate a valid hash value (cf. Section 2.3).

HMAC is one of the key dependent one way hash functions. It is widely used in protocols like TLS or IPsec. Its specification can be found in [31]. Within the creation of an HMAC tag, an unkeyed hash function has to be used. A common implementation is HMAC-SHA which employs SHA as embedded unkeyed hash function.

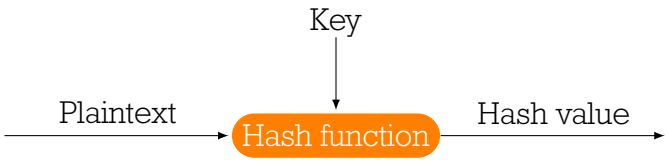


Figure 2.10: Keyed hash function

A main advantage of HMAC is, that the underlying hash function has not to be modified. Well established and fast hash functions can be used and – if necessary – exchanged easily.

The creation of the HMAC-tag takes 2 bit wise XOR operations and two hashing operations $H(x)$. The operator $||$ denotes the concatenation of two strings.

$$HMAC(SK_{MAC}, txt) = H\left((SK_{MAC} \oplus opad)\,||H\left((SK_{MAC} \oplus ipad)\,||txt\right)\right)$$

SK_{MAC} is the shared secret key for the Message Authentication Code (MAC) calculation. The secret key SK_{MAC} is XOR'ed with the constant value $ipad = 36_h$ and concatenated with the message text txt. After applying the hash function H (like SHA-256), the result will be appended to the key SK_{MAC}, XOR'ed with the constant $opad = 5C_h$. The application of the hash function H to the whole string calculates HMAC tag.

Since the output of the used hash function is the output of the HMAC operation, the length of the HMAC tag depends on the used hash function. E.g. when using SHA-256, the output of HMAC is 32 bytes long. The shared secret key should not be smaller than the output of the hash function. There is no upper size limit for the key length, but keys which are longer than the block size of the hash function, will be hashed once to reduce the size.

Appendix C, Listing C.1 comes up with an implementation of the HMAC function.

2.5.2 Digital signature

A digital signature is typically shipped together with the message it is related to. Basically it claims that the message has not been altered (integrity) and authenticates the sender (entity authentication). The following enumeration lists the properties of signatures (cf. [11]):

Authentic. The signer deliberately signed the document.

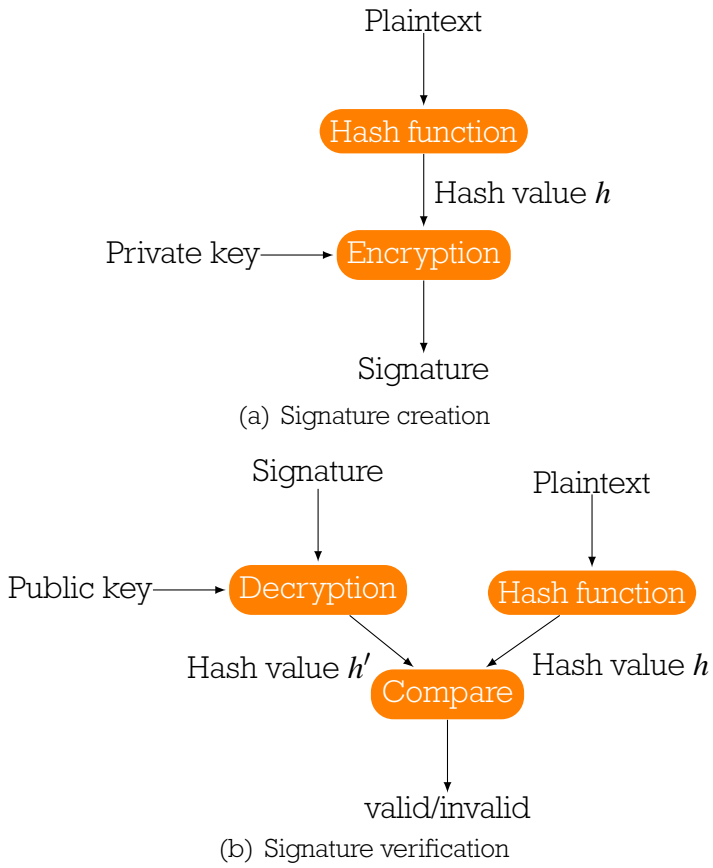
Unforgeable. No one else than the signer signed the document.

Not re-usable. The signature is related to the document. It is not possible to use this signature with other messages.

Unalterable. The document has not been changed (by anyone) after the signature has been created.

Non repudiation. The signer cannot deny, that he did not sign the document.

While handwritten signatures, which are still used in a wide field of applications, are not able to fulfill all of these properties, digital signatures can.



(a) Signature creation

(b) Signature verification

Figure 2.11: Digital signature

The signatures are generated using asymmetric cryptographic algorithms and hash functions. The basic functionality of common digital signature algorithms is shown in Figure 2.11. During signature generation (cf. Figure 2.11(a)), a hash function is used to reduce the amount of data to a fixed length for the encryption. The resulting hash value

is encrypted with the private key of the sender. The encrypted hash value represents the digital signature and is transmitted together with the message. Because of the use of a hash function to calculate a message hash, the digital signature inherits the integrity property of the hash functions.

To validate the digital signature (cf. Figure 2.11(b)), the received signature is decrypted using the public key of the sender. The decrypted signature is the reference hash value h'. The hash value h is calculated by applying the hash function on the message. If the hash value h and the reference h' are the same, the signature is valid. Otherwise the signature is invalid.

Like asymmetric encryption algorithms, the digital signature algorithms can be categorised by the underlying mathematical problem.

IFP

The first class of digital signature algorithms is based on the IFP. The signature scheme related to the RSA encryption (cf. Section 2.4.2) is shown exemplary.

RSA signature scheme Besides encryption, RSA can be used to sign a message. The key pair generation remains the same compared to the encryption, as already described in Section 2.4.2. To generate the signature s, the sender uses its private key to encrypt the hash of the message, which is the output of a (cryptographic) hash function $H(m)$ (cf. Section 2.5.1):

$$h = H(m) \tag{2.17}$$

$$s \equiv h^d \mod n \tag{2.18}$$

The signature s is transmitted together with the plaintext (or encrypted) message m to the receiver. The receiver will also calculate the message hash h via the Equation 2.17. Afterwards the signature will be decrypted using the public key of the sender, to get the reference hash h':

$$h' \equiv s^e \mod n \tag{2.19}$$

According to the conversions on Page 17, it is easy to show that $h = h'$, if the signature s and the message fingerprint h have not been modified:

$$\begin{aligned} s^e \mod n &= h^{ed} \mod n \\ &= h^{1+k\phi(n)} \mod n \\ &= h(h^k)^{\phi(n)} \mod n \\ &= h \mod n \end{aligned}$$

So, if the equation $h = h'$ holds, the signature is valid and the receiver can be sure that the sender knows the private key associated to the used public key and that the message was not modified.

DLP

As representatives of the DLP based digital signature algorithms, the ElGamal signature scheme and the Digital Signature Algorithm (DSA) are presented.

ElGamal Signature scheme As already mentioned, the ElGamal algorithm can also be used to create and verify digital signatures on messages. For the generation of the keys, the equations of Section 2.4.2 are used[2].

If Alice wants to send a signed message to Bob, again Alice has to choose a random number k (with the same requirements stated in the Equations 2.11 and 2.12). The pair (r,s), which represents the signature, can be calculated via the following equations

$$r \equiv g^k \mod p \tag{2.20}$$

$$s \equiv (H(m) - x \cdot r)\, k^{-1} \mod (p-1) \tag{2.21}$$

where the used function $H(m)$ is a (cryptographic) hash function (cf. Section 2.5.1).

The usage of the value k is subject to the same limitations as stated in Section 2.4.2.

[2] x denotes the private key of Alice and (y,g,p) her public key

To verify the signature, Bob has to check, if the following equation holds (using the public key y of Alice).

$$g^{H(m)} \equiv y^r \cdot r^s \mod p$$

With the help of the introduced equations, the following transformation shows the correctness of the signature scheme:

$$\begin{aligned} H(m) &\equiv x \cdot r + k \cdot s \mod (p-1) \quad (\rightarrow \text{Eqn. 2.21}) \\ g^{H(m)} &\equiv g^{x \cdot r + k \cdot s} \\ &\equiv g^{x \cdot r} g^{k \cdot s} \\ &\equiv (g^x)^r (g^k)^s \\ &\equiv y^r \cdot r^s \mod p \quad (\rightarrow \text{Eqn. 2.10 and Eqn. 2.20}) \end{aligned}$$

Digital Signature Algorithm (DSA) The DSA has been announced by the NIST in August 1991. The intention for this standard was the establishment of a secure public key digital signature standard, which can be used for federal digital signature applications. Its algorithm is a variant of the ElGamal signature algorithm, described in Section 2.5.2. A full description of DSA can be found in [32].

Keypair generation The keypair generation of DSA is similar to the one of ElGamal. First, one entity has to choose a prime number p. Depending on the desired level of security, the bit length of this number ranges from 512 up to 3072 bits and is denoted by L. The next number q has to be a prime factor of $p-1$. The length of q (denoted with N) is specified by [33] and depends on L. The following pairs of (L,N) are allowed: $(1024,160)$, $(2048,224)$, $(2048,256)$ and $(3072,256)$. The third "random" number is g, calculated via

$$g = h^{\frac{(p-1)}{q}} \mod p \tag{2.22}$$

h is chosen in a way, that $g > 1$. The triple (p,q,g) is the domain parameter and is public. According to ElGamal, the private key x is randomly chosen and the public key y is determined by Equation 2.10.

Signature To sign a message, a hash function $H(m)$ is used to create a hash value out of the plaintext message. The values r and s, which represent the digital signature of the message, are calculated via the following equations:

$$r \equiv (g^k \mod p) \mod q \tag{2.23}$$
$$s \equiv (H(m) + x \cdot r)\, k^{-1} \mod q \tag{2.24}$$

Since the value of r does not depend on the message itself, a precomputation can be done to speed up the creation of the signature. Also the value of k^{-1} can be calculated in advance. Detailed description of these optimisations and measurements can be found in [11].

Verification To verify the signature, the receiver has to do the following calculations:

$$w \equiv s^{-1} \mod q \tag{2.25}$$
$$u_1 \equiv H(m) \cdot w \mod q \tag{2.26}$$
$$u_2 \equiv r \cdot w \mod q \tag{2.27}$$
$$v \equiv (g^{u_1} y^{u_2} \mod p) \mod q \tag{2.28}$$

If $v = r$, the signature is valid. The correctness of the algorithm is shown as follows:

$$\begin{aligned} k &= \frac{H(m)}{s} + \frac{xr}{s} \quad (\rightarrow \text{Eqn. 2.24}) \\ &= H(m)w + xrw \quad (\rightarrow \text{Eqn. 2.25}) \\ &= u_1 + u_2 x \quad (\rightarrow \text{Eqn. 2.26 and Eqn. 2.27}) \\ r = g^k &= g^{u_1} g^{u_2 x} \\ &= g^{u_1} y^{u_2} = v \quad (\rightarrow \text{Eqn. 2.10}) \end{aligned}$$

ECDLP

The following digital signature scheme is based on the ECDLP, which has been introduced in Section 2.4.2.

Elliptic Curve Digital Signature Algorithm (ECDSA) Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve pendant to the DSA, which has been discussed in Section 2.5.2. It is an International Organisation for Standardisation (ISO) standard since 1998 (ISO 14888-3). Among others, its specification can be found in [34].

Keypair generation The elliptic curve key pair is associated with a particular domain parameter D. To generate a key, a random number d has to be chosen, which represents the private key. The public key is a point on the elliptic curve:

$$Q = dP \tag{2.29}$$

Signature In the scenario, Alice wants to send a digitally signed message to Bob. The formulas for ECDSA are very similar to the ones of DSA. The input parameters of the ECDSA are the domain parameter $D = (q, \mathrm{FR}, S, a, b, P, n, h)$, Alice's private key d and the message m. First, Alice has to choose a random number k. With this number, she can calculate the signature[3]:

$$(x_1, y_1) = k \cdot P \tag{2.30}$$

$$r \equiv \overline{x_1} \mod n \tag{2.31}$$

$$s = k^{-1}(H(m) + d \cdot r) \tag{2.32}$$

The pair (r, s) is the digital signature, which will be used by Bob to validate the transmitted message.

[3] $\overline{x_1}$ is the integer representation of x_1

Verification To verify the digital signature, Bob needs the domain parameter $D = (q, \mathrm{FR}, S, a, b, P, n, h)$, Alice's public key Q, the message m and the signature (r, s):

$$w \equiv s^{-1} \mod q \tag{2.33}$$
$$u_1 \equiv H(m) \cdot w \mod q \tag{2.34}$$
$$u_2 \equiv r \cdot w \mod q \tag{2.35}$$
$$(x_2, y_2) = u_1 P + u_2 Q \tag{2.36}$$
$$v \equiv \overline{x_2} \mod n \tag{2.37}$$

These formulas are almost the same compared to the ones of DSA. If $v = r$, the signature is valid. The correctness of the algorithm is shown as follows (note: $(x_1, y_1) = (x_2, y_2) \Rightarrow v = r$):

$$\begin{aligned} k &= \frac{H(m)}{s} + \frac{dr}{s} \quad (\rightarrow \text{Eqn. 2.32}) \\ &= H(m)w + drw \quad (\rightarrow \text{Eqn. 2.33}) \\ &= u_1 + u_2 d \quad (\rightarrow \text{Eqn. 2.34 and Eqn. 2.35}) \\ (x_1, y_1) = kP &= u_1 P + u_2 dP \quad (\rightarrow \text{Eqn. 2.30}) \\ &= u_1 P + u_2 Q = (x_2, y_2) \quad (\rightarrow \text{Eqn. 2.29}) \end{aligned}$$

2.6 Freshness

Data freshness ensures that an arriving message was not received (and processed) up to now. If a message arrives more than once – e.g. because of network retransmission or because of a replay attack – this will be recognised and the message is discarded. Data freshness can be achieved by adding a unique identifier to the message (cf. Figure 2.12). This can be done e.g. by embedding a sequence counter into the message. Since the sender changes this counter every time a message is sent (e.g. by increasing it by 1), it can be used to guarantee data freshness. The receiver extracts the unique identifier and checks (e.g. via a lookup table), if this identifier has already been used. The identifier itself (or the whole message) has to be immutable (e.g. by means of integrity protection – cf. Section 2.5).

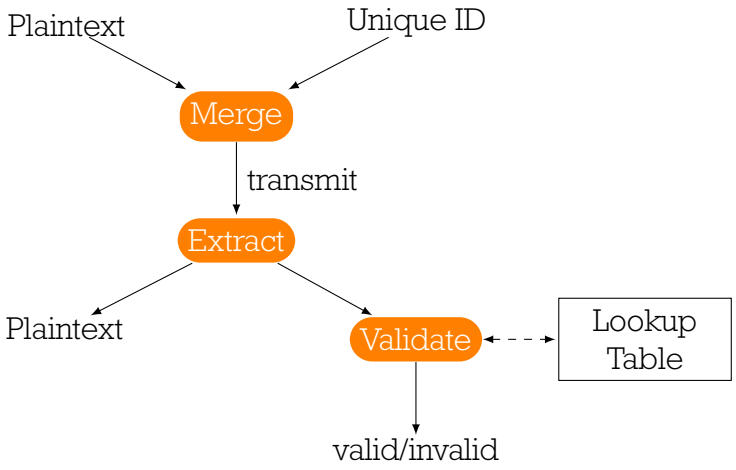


Figure 2.12: Data freshness

2.7 Elliptic Curve Integrated Encryption Scheme (ECIES)

Hybrid algorithms combine the easier key distribution capabilities of asymmetric algorithms with the high operation speed of symmetric ones. Because of these advantages, most of the common cryptographic protocols and schemes are hybrid. This section explains one hybrid encryption scheme. State of the art protocols which use symmetric as well as asymmetric algorithms can be found in Section 3.

ECIES is based on the ElGamal algorithm and provides a secure method to exchange messages like short keys for symmetric encryption. Among others, the SECG provides a detailed specification of ECIES (cf. [35]).

The data is secured with a symmetric algorithm together with an authentication scheme using a hash algorithm. The keys for these algorithms are derived with Elliptic Curve Diffie-Hellman (ECDH) key exchange (cf. Section 2.9.3).

The full scheme is divided into 4 parts. First, in the "Scheme Setup", the used algorithms like the Key Derivation Function (KDF), the encryption algorithm and the MAC algorithm, as well as ECC domain parameters are agreed. During the second part, called "Key Deployment", the receiver has to gain the sender's public key in an authentic manner. The "Encryption Operation", which is the third part of the encryption scheme, is visualised in Figure 2.13. An ephemeral elliptic curve key pair has to be created. The newly created private key and the public key of the receiver are used as

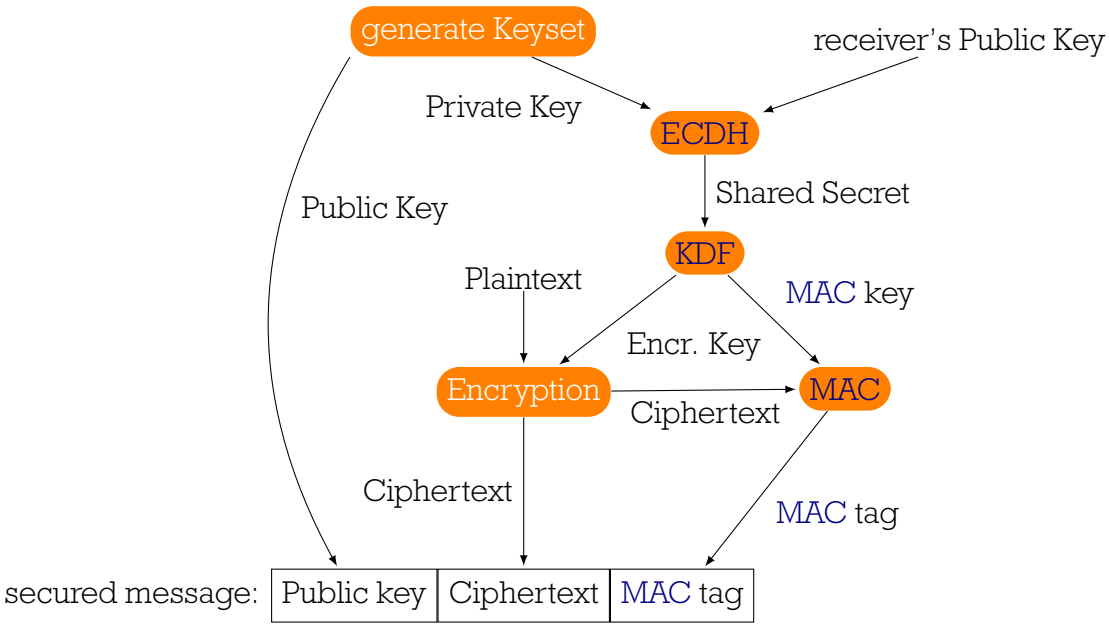


Figure 2.13: ECIES encryption

input to a ECDH routine to generate a shared secret (cf. Appendix C, Listing C.2 for an example implementation and a description of this function). The negotiated KDF is applied to the shared secret to calculate the encryption key and the MAC keys. The encryption operation calculates the ciphertext out of the plaintext, using the calculated encryption key. The MAC algorithm calculates the MAC tag, using the calculated MAC key. The sender sends the temporary public key concatenated with the encrypted payload and the MAC tag to the receiver.

In the last part, the ''Decryption Operation'', (cf. Figure 2.14) first of all the public key and the MAC tag are extracted from the whole message. Like during the ''Encryption Operation'', a DH routine is used to regenerate the shared secret. The inputs of this routine are the private key of the receiver and the received temporary public key. The KDF produces the needed keys for the MAC verification and decryption.

2.8 Randomness

The security of most of the cryptographic algorithms is based on the quality of random numbers. These random numbers are used to generate initial secrets, which are only known to the owner of this random number. In other words: the best cryptographic

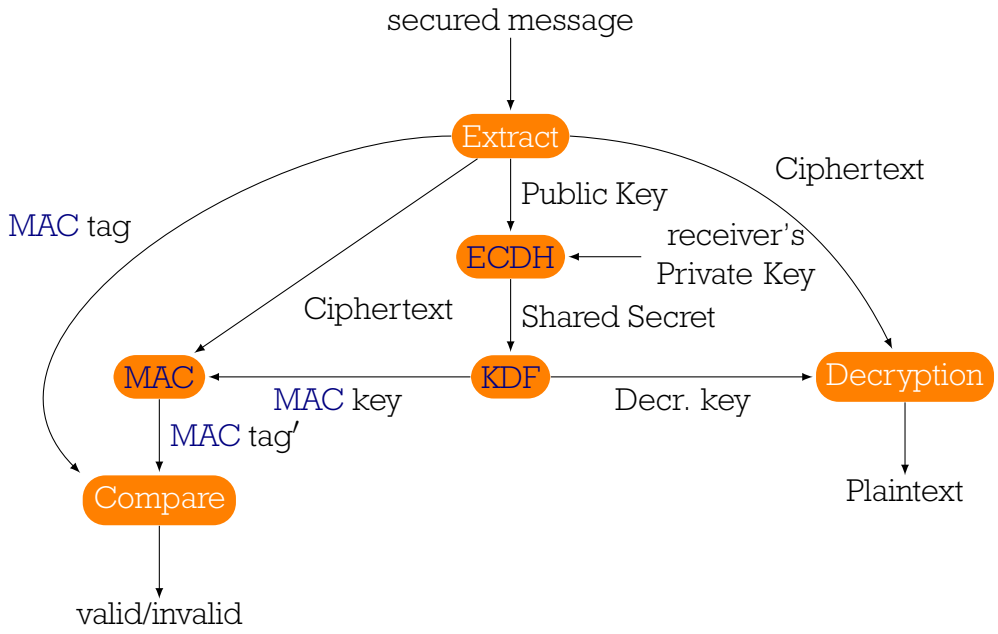


Figure 2.14: ECIES decryption

algorithm can be broken, if the underlying random numbers are known or predictable. The main requirements (according to [36]) to random generators are

Randomness. The output is random with a uniform distribution.

Forward security. If an attacker knows the internal state of the generator, he is not able to calculate previous outputs.

Backward security. The attacker is not able to calculate future outputs of the generator.

Computers are expected to be completely deterministic. The same input produces the same output – at any time. The finite number of possible internal states limits the possibilities of the output. Any random number generator on a computer is per definition periodic. Therefore generating a random number is not just a difficult task for computers – it's impossible. With the words of John von Neumann: ''Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.''

The best output of a computer can be a pseudo random sequence. In software, these pseudo random numbers are generated with a Pseudo Random Number Generator (PRNG). Depending on the field of application of the output, different quality criteria may apply. According to the properties listed in [11], PRNGs are said to be good, if they look random. I.e. they have to pass several statistical tests of randomness, like

the uniform distribution. Cryptographically secure PRNGs additionally have to be unpredictable, even if the whole previous output and the algorithm are known. Since computers are deterministic, PRNGs have to be initialised with some random number in order to produce different output sequences in different executions. This initial value is called seed. Usually some kind of external event (e.g. keystrokes, time, message arrival interval, ...) is used to generate the seed.

Many libraries, operating systems and compilers provide different implementations for PRNGs. But the quality of these "standard" implementations do not fulfill the requirements for cryptographic applications. The generated numbers appear to be random, but are often completely determined by a small seed. Furthermore the unavoidable period is often very small.

The following example shows an easy implementation of a PRNG, the Linear Feedback Shift Register (LFSR) to demonstrate predictability and the period.

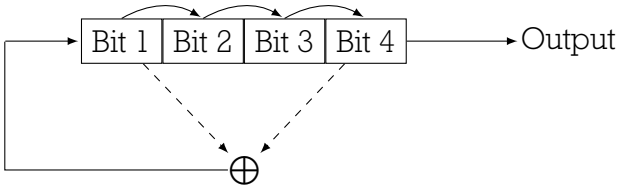


Figure 2.15: LFSR example

In a LFSR, several bits inside the shift register are combined with a logical feedback function to form the next input of the shift register. The output represents the pseudo random bits. The example in Figure 2.15 uses an XOR as feedback function, taking bit 1 and bit 4 as inputs. Within every step, the content of bit 4 becomes the new output, the content of bit 3 is shifted to bit 4, bit 2 is shifted to bit 3 and so on. The result of the XOR operation is used as new value for bit 1.

With a seed of 12 (1100_b), the output sequence shown in Table 2.2 will occur.

Bit 1	Bit 2	Bit 3	Bit 4	XOR result	output
1	1	0	0	1	
1	1	1	0	1	0
1	1	1	1	0	0
0	1	1	1	1	1
1	0	1	1	0	1

0	1	0	1	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	0	1	0	1
0	1	1	0	0	1
0	0	1	1	1	0
1	0	0	1	0	1
0	1	0	0	0	1
0	0	1	0	0	0
0	0	0	1	1	0
1	0	0	0	1	1
1	1	0	0	1	0

Table 2.2: LFSR example

The output of the LFSR in Figure 2.15 seems to be completely random. But a closer look to it reveals a period of 15. So even a brute force attack, which tries all possible initial values will succeed very fast.

While no one would use such a simple PRNG for cryptographic applications, even well established and often used generators have threatening weaknesses. [36] shows attacks against the most commonly used PRNG, the one of the Windows operating system (Windows 2000). Another huge security flaw has been found in an OpenSSL implementation of Debian based operating systems. The implementation used a very small "random" source as seed for the PRNG (cf. [37]). Besides the limited range of the entropy source, many of the possible initial values are very unlikely and may not be taken into consideration when attacking such a system.

Since PRNGs are used inside cryptographic algorithms and to generate keys, for cryptanalysts PRNGs are as interesting as the cryptographic algorithms themselves. To break the security of a system, one could either find a vulnerability of the used cryptographic algorithm or of the deployed PRNG.

Several documents describe the threats of PRNGs and propose secure implementations and good random sources. E.g. in [38] approved algorithms and testing tech-

niques of the NIST are stated. [39] focuses on random sources and introduces pseudo random number generation processes.

2.9 Key management

Keys are a central entity in cryptographic systems. Except unkeyed hash functions, all of the described algorithms need one or more keys to work. Most of these keys have to be secret and should only be available to authorised parties.

Key management covers several tasks (cf. [13]). One part of key management is the process to establish a shared secret key, called key establishment. This includes the generation and distribution of the keys. There are three groups of key establishment algorithms:

Pre-shared or manually shared key. The user has to explicitly share secrets.

Symmetric key technique. A third trusted party has to be involved.

Asymmetric algorithms.

It is very common, that algorithms of these groups are executed consecutively. Especially pre-shared keys are necessary to execute symmetric or asymmetric key techniques. Another main part of key management is the control of the keying material as well as the update, revocation and destruction of keys. Basically, there are two situations where the revocation of a key set and the distribution of a new one are required (often referred to as rekeying): First, it may be possible that a communication member has to be actively excluded from a communication relation. This may be necessary if a device becomes compromised. Second, due to security reasons, an active key set shall be recreated periodically to reduce the number of key set uses (and to avoid an overflow of a possible sequence counter[4]). The key storage and the backup/recovery mechanisms are also part of the key management.

2.9.1 Pre-shared

Cryptographic systems with a pre-shared key exchange the keys in advance in a secure manner. The system itself does not care about the key exchange – the keys have

[4] A sequence counter is used to ensure data freshness (cf. Section 2.6)

to be available on all participated entities when the secured communication starts.

The keys have to be exchanged via some secure channel. Since usually no other security mechanisms are available at that time, a physically secured channel has to be used. Typically, a user who has a direct connection (e.g. via EIA-232 or USB) to the participating entities will distribute the keys. This can be done e.g. by entering a pass-phrase at the devices or by exchanging stored keys via transportable mediums like USB sticks. If the keys are not exchanged via network connections, it is almost impossible for an attacker to gain the shared secret during the key exchange process.

A big drawback of pre-shared keys is the fact, that the key exchange is not very comfortable. It is long-winded and maybe expensive to renew the used keys. Since in some systems the user has to enter a password or a pass-phrase, the used keys may be insecure because of poor passwords (cf. [13]).

Usually, pre-shared keys are used as basis for symmetric or asymmetric key exchange techniques. Since it is almost impossible to do automatic rekeying, pre-shared systems are not very practicable. But there are some application fields, where pre-shared keys are still used and offer a great level of security. In electronic banking systems Transaction authentication numbers (TANs) are used to transfer money in a secure manner. The TAN lists, which are distributed via regular mail or the bank office, are a form of pre-shared keys, which are used only once (one-time password).

2.9.2 Symmetric

A possible solution for symmetric key management is shown in Figure 2.16. The keys are exchanged via a TTP called Trent in the example (cf. [13]). In conjunction with symmetric encryption keys, the TTP is also called key server. Each participant shares a distinct symmetric key with the TTP. These keys have to be distributed via a secured channel in advance (cf. Section 2.9.1).

If Alice wants to talk to Bob in a secure manner, they will contact Trent with the help of the symmetric keys K_{AT} and K_{BT}, which have been distributed in advance. Trent will generate a session key k and send this key to Alice and Bob via the secured communication channel (the feedback channels are also secured using K_{AT} and K_{BT}). Alice and Bob can use the session key k, which is only known by them (and Trent), to establish a

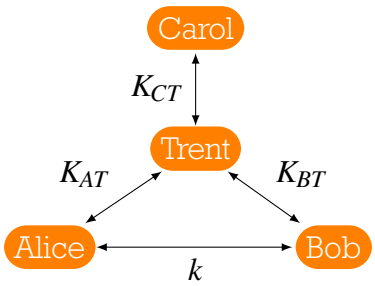


Figure 2.16: Symmetric key management

secured channel between them. A noninvolved party like Carol or an attacker Eve will not be able to gain the session key k.

This example using a TTP can be compared to the Kerberos protocol (cf. [40]). One advantage of this approach is, that it is easy to add and remove entities from the network. This process can be entirely controlled by the TTP. Another advantage is, that each entity has to store only one long-term secret key.

But there are also several disadvantages of this concept. Every communication requires an interaction with an online TTP which probably puts high load on it and slows down the establishment of secured channels. The TTP has to store several long-term secret keys – one for each entity. Since the TTP knows the session keys, it is able to read all messages. If the TTP is compromised, the whole communication within the network becomes insecure. This centralised approach also implies, that a failure at the TTP will stop the establishment of new connections in the whole network.

2.9.3 Asymmetric

Asymmetric key exchange mechanisms use asymmetric cryptographic algorithms to establish a shared key between two or more parties. Each participating party has it's own private/public keypair, where only the public one has to be shared among all parties. With the knowledge of the public keys and the own private key, a shared secret can be calculated. Since an eavesdropper only knows the public keys, but not the private keys, it cannot calculate the shared secret.

Certification Authority (CA)

A TTP, which in this context is also called CA, creates certificates, which prove the identity of an entity. A digital certificate consists of a public key and some kind of identifying information, which later on can easily be verified (like the address). The two parts of the certificate are bound together with a digital signature (cf. Section 2.5.2), created by the CA.

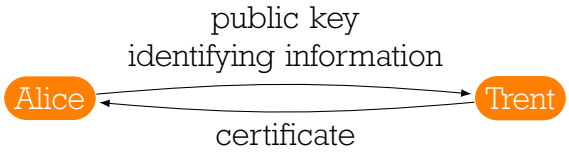


Figure 2.17: Certificate creation

If an entity of a security domain wants to gain a certificate, it has to provide it's public key and prove it's identity (cf. Figure 2.17). The CA verifies the identifying information and creates the certificate, if the verification was successful: it digitally signs the certificate with a negotiated digital signature algorithm (cf. Section 2.5.2) and ships the whole certificate back to the applicant. The communication channel for this initial communication between the requesting entity and the CA must provide entity authentication, integrity and freshness. Confidentiality is not necessary.

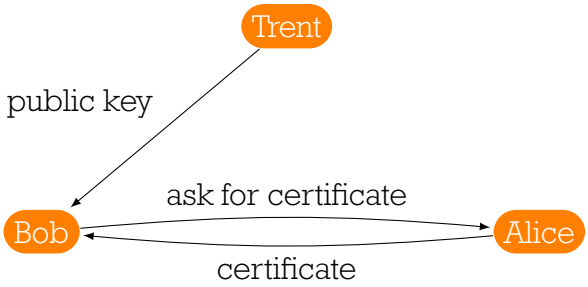


Figure 2.18: Certificate verification

If Bob wants to verify, if the communication partner really is Alice, he can ask for Alice's certificate and verifies it (cf. Figure 2.18). To verify the certificate, Bob has to verify the digital signature of the TTP, which is included in the certificate, by the means of digital signature algorithms (cf. Section 2.5.2). To do this, Bob only has to know the CA's public key, which has to be shared in advance (cf. Section 2.9.1). If the signature is correct, Bob knows that Trent checked the identity of Alice and can trust Alice's identity.

Diffie-Hellman (DH)

DH was the first public key algorithm ever. It was published in 1976 by Whitfield Diffie, Martin Hellman and Ralph Merkle and is based on the DLP (cf. [11]).

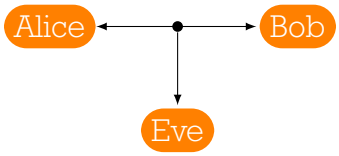


Figure 2.19: DH principle

A basic scenario is shown in Figure 2.19. Assume, that Alice and Bob want to establish a shared secret. First, both parties have to agree on a random prime number p and the primitive root[5] g modulo p. Now, Alice chooses a random large integer x and calculates

$$X \equiv g^x \mod p$$

Accordingly, Bob also chooses some random large integer y and calculates

$$Y \equiv g^y \mod p$$

After the values of X and Y are exchanged, Alice will calculate k_A and Bob will calculate k_B

$$\begin{aligned} k_A &\equiv Y^x \mod p \\ k_B &\equiv X^y \mod p \end{aligned}$$

The values k_A, calculated by Alice, and k_B, calculated by Bob, are the same, since

$$k_A \equiv g^{xy} \mod p \equiv k_B$$

No other party listening on the channel (like Eve) and intercepting p, g, X and Y is able to calculate the value of k in reasonable time, because of the DLP (cf. Section 2.4.2).

If the values for p and g are chosen properly, the random numbers x and y are of good

[5] The primitive root g of a cyclic (finite) group is the generator of this group. I.e. the generator multiplied with itself (multiple times) generates all elements of this group.

quality (cf. Section 2.8) and the communication channel provides integrity protection, this protocol is considered to be secure. Hence, the DH key exchange algorithm provides a mechanism to derive a common secret, if an attacker has only the possibility to intercept messages but is not able to modify messages.

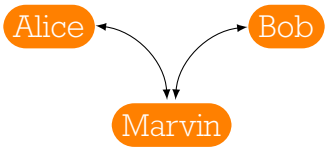


Figure 2.20: DH man-in-the-middle

If the attacker is able to alter the sent messages, DH is not secure anymore. Figure 2.20 shows a typically scenario, where the attacker Marvin can modify the messages of Alice and Bob. During the communication with Alice, Marvin pretends to be Bob. From Bob's point of view, Marvin seems to be Alice. Marvin executes the DH algorithm twice to establish two shared secret keys. One key is shared with Alice, the other one with Bob. This kind of attack is called man-in-the-middle attack.

Since the DH protocol is not authenticated, Alice and Bob will not recognise the difference. If entity authentication is necessary, the Station-to-Station protocol can be used.

To save memory consumption and computational power, elliptic curve primitives can be used to execute the key agreement. These algorithms which are also suitable for constrained environments are called ECDH.

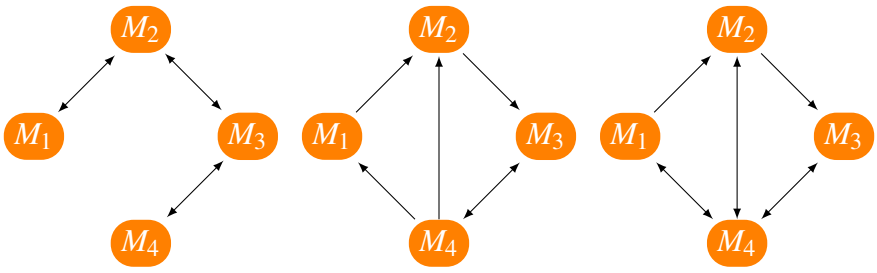
Station-to-Station Protocol

The Station-to-Station Protocol (cf. [11]) solves the man-in-the-middle vulnerability of the DH protocol by introducing an authentication mechanism. The entity authentication is done with the help of certificates. Alice has Bob's certificate with Bob's public key and Bob knows Alice's certificate. The certificates have to be signed by a TTP outside this protocol (cf. Section 2.9.3).

Basically, the messages between Alice and Bob are digitally signed. The signatures can be verified with the aid of the certificates. Since an adversary is not able to generate valid signatures, the man-in-the-middle attack is not possible anymore.

Group Diffie-Hellman (GDH)

DH only supports the secure key agreement between two parties. To agree on a common key within a group, several group key distribution protocols are available. [41] presents and analyses the modification of the 2-party DH algorithm for groups to form the GDH algorithms. GDH.1, GDH.2, and GDH.3 have different fields of application. GDH.1 does not need any special communication requirements like broadcasting or synchronisation. There is only point-to-point communication, as shown in Figure 2.21(a). GDH.2 is optimised with respect to the number of rounds. The necessary communication lines are shown in Figure 2.21(b). Key distribution with a low amount of exponentiations is done by GDH.3. Like GDH.2, it needs an almost fully connected or broadcasting network in order to work (cf. Figure 2.21(c)). All of the these algorithms can also use the ECDH key exchange in place of the regular DH to save computational power and lower memory consumption (cf. [25, 24]). Since computational power is the main limiting factor on embedded devices, [42] presents a modified version of the GDH.3 protocol, which additionally saves some operations.



(a) GDH.1 (b) GDH.2 (c) GDH.3

Figure 2.21: GDH network requirements

Exemplarily, this modified GDH.3 algorithm is shown here. The n members of the group are arranged in a logical chain beginning at member M_1, ranging to member M_n. First, every member M_i chooses a random secret k_i. This number is the private key of the i-th member. M_1 selects some elliptic curve point P and sends the value $Q_1 = k_1 P$ to M_2. Q_1 is the public key of M_1. M_2 continues by sending the value $Q_2 = k_2 Q_1 = k_2 k_1 P$ to M_3. When reaching the last member M_n, it encrypts the group key Q_n with the public key Q_{n-1} of its predecessor M_{n-1}. M_{n-1} can decrypt the received message with its private key k_{n-1} to obtain the secret value Q_n. Afterwards, it encrypts Q_n with the public key Q_{n-2} of its predecessor M_{n-2}, and so on.

As a result of this protocol, every group member knows the secret value Q_n. Every member has to perform one point multiplication. All but one members have to encrypt and decrypt the secret value one time. Table 2.9.3 compares this result with the other GDH protocols.

protocol	point multiplications	encryption[6]	decryption[6]
GDH.1	$i+1$ for $i<n$, n for $i=n$	0	0
GDH.2	$i+1$ for $i<n$, n for $i=n$	0	0
GDH.3	4 for $i<n-1$, 2 for $i=n-1$, n for $i=n$	0	0
GDH.3 modified	1 $\forall n$	1 for $i<n$, 0 for $i=n$	1 for $i>1$, 0 for $i=1$

Table 2.3: Comparison of group key distribution protocols

One problem of these protocols is, that the members have to be ordered and the number of members in the group has to be known. Besides, since these GDH protocols are non-authenticated, they inherit the problem of the man-in-the-middle attack from the DH algorithm. Again, this can be overcome by the use of some entity authentication mechanism.

2.10 Securing multicast

Multicast is an efficient way to send a message to more than one receiver. Instead of sending the same message several times, the sender will send it out once and all intended receivers will receive the message (cf. Figure 2.22). This saves traffic bandwidth and processor load at the sender. Depending on the network topology (e.g. star topology), it is possible that the multicast message will be cloned several times within the used network by the network infrastructure (e.g. routers) to reach all receivers.

There are basically three ways to encrypt multicast traffic.

[6] asymmetric with ECC algorithms

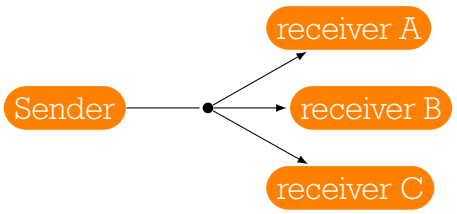


Figure 2.22: Multicast

Multicast as many unicasts. A multicast can be simulated as many unicasts. Every sender maintains a point-to-point connection to every receiver. If every member is allowed to send messages, every member has to share one key with every other member. From the key management's point of view, with increasing group size this method becomes very inefficient during setup and operation. It scales very bad and needs a lot of memory. Furthermore this way of multicast encryption is very slow during operation because each sender has to encrypt every message several times – once for every other group member.

Centralised with server. Every group member maintains a point-to-point connection to the server. Each multicast message will be routed via the server. The server decrypts the message from the sender and sends the encrypted message to all other group members. In contrast to the previous method, every group member has to store only one key and the senders have to encrypt the message only once. But the server needs high network bandwidth and computational power. Besides, this server represents a single point of failure.

Single group key. All members of a multicast group share a common secret key. Only the members know this key and can therefore en- and decrypt the messages. A sender has to encrypt a message only once and every member can decrypt this message. During operation, this is the most effective way to secure multicast. The main problem is the efficient key management. If the group is dynamic the key has to be changed (referred to as rekeying) if new members are joining the group or some members are leaving the group.

Chapter 3

Security for IP backbones in HBA

Due to the widespread use of the Internet, security has been a major research field in the IT world for years. Therefore, many different security mechanisms for IP based networks are available. Each of them is suitable for a certain application field. In this section, the most important IT mechanisms are analysed with respect to their suitability for KNXnet/IP.

Since IT world comes up with several security protocols to secure IP traffic, it would be obvious to use already specified, implemented and well-established products to satisfy the requirements stated in Section 1.3.

This section compares the most popular protocols, which come into consideration for KNXnet/IP, and sums up the results at the end.

3.1 Internet Protocol Security (IPsec)

IPsec is an extension in the IP specification. When using IPsec, it replaces IP – cf. Figure 3.1. It is a part of Internet Protocol Version 6 (IPv6) but since Internet Protocol Version 4 (IPv4) is still the predominantly used network protocol in the IT world, it has been ported to extend IPv4. The implementation of IPsec in IPv6 is mandatory, while the implementation within IPv4 is optional. Its specification is spread over several RFCs, ranging from RFC4301 to RFC4309, available at [43].

IPsec can ensure mutual entity authentication and may provide data integrity, freshness, and confidentiality. It supports different cryptographic algorithms for encryption

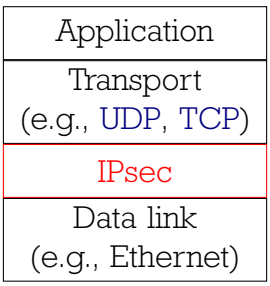


Figure 3.1: Position of IPsec in the OSI layer model

and signature generation. For example, Data Encryption Standard (DES), Triple-DES, or AES can be used as encryption algorithm and HMAC-SHA1 for MAC calculation. For key exchange, again IPsec is able to use a bunch of mechanisms and protocols. Besides the manual distribution of keys, mainly the IKE protocol is used.

3.1.1 Security Association (SA)

A basic concept in the IPsec specification is the SA, which is always unidirectional and connects two communication endpoints. A two-way connection needs therefore at least two SAs. The SA describes how the communication is secured and which features are enabled. It manages the necessary keys, defines the used algorithms and possible filter policies among other parameters.

The two main protocols within IPsec are AH and ESP. While AH only provides authentication and data freshness, ESP additionally provides confidentiality.

3.1.2 Authentication Header (AH)

When using AH, the authentication is done via a hash value calculated over the entire IP packet. Only few fields of the IP header record – like the Time To Live (TTL) or header checksum – are excluded from this calculation. A monotonically increasing sequence number inside the AH header is used to avoid replay attacks and ensure data freshness. The AH header precedes the payload – it is situated between the IP header and the header of the succeeding layer (e.g. the Transmission Control Protocol (TCP) header). The main drawback of AH is the consideration of the IP source and destination address during the calculation of the authentication data. Therefore these fields are immutable,

and AH cannot be used with Network Address Translation (NAT) routers, which have to alter these fields.

3.1.3 Encapsulated Security Payload (ESP)

In ESP, authentication is optional. But since the encrypt-only usage of ESP is subject to effective attacks (cf. [44]), the usage of ESP without authentication should be avoided. If it is turned on, again a hash value is calculated to build the authentication data. Unlike AH, ESP only uses the ESP header and the encrypted payload for the calculation – so there are no problems behind NAT boxes. Like in the AH header, a sequence number is used for replay protection. Because of the encryption, ESP has to surround the payload and appends a trailer (padding, authentication data, ...) to it.

3.1.4 Modes of operation

There are two operation modes for IPsec. Both AH and ESP can be used in transport or tunnel mode. When using transport mode, a secure connection between two endpoints can be established and IPsec encapsulates the IP payload. The tunnel mode encapsulates the whole IP packet including the IP header and can be used to establish Virtual Private Networks (VPNs) (cf. Section 3.3).

3.1.5 Internet Key Exchange (IKE)

As already mentioned, IKE is used for key exchange. While the first version of IKE had problems with NAT traversal and firewalls, these problems are solved with the second version IKEv2 (which will be discussed here – IKEv2 and IKE are used synonymously). The IKE performs the requested mutual authentication between two parties and is used to establish an IKE SA which includes shared secret information. Out of the IKE SA an IPsec SA can be created which will be used later on for ESP and/or AH.

Every IKE request message is answered by a response message of the other party. In the common case, 4 messages are needed to establish an IKE SA. The initiator starts with an `IKE_SA_INIT` message, where security parameters, nonces, available

cryptographic suites and DH values are set. The responder chooses one suite, completes the DH handshake and sends a new nonce. After this response, the encrypted and authenticated (integrity protection) request/response `IKE_AUTH` messages proves knowledge of the secrets and authenticates the two parties. Within this pair of message, an IKE SA can be established.

IKE is strongly related to the DH key exchange. According to the specification [45], the extensible framework is able to cope with elliptical curve cryptography for a higher level of security and a faster execution. But still, the DH algorithm has to be used.

3.1.6 IPsec and multicast

IPsec is intended to secure both unicast and multicast communication. To secure multicast, one SA is created for a whole group. The sender to a multicast group uses this SA to encrypt data in symmetric manner. Without any extensions, the receivers are not able to authenticate the senders of the messages. However, there are some problems when multicast is used in IPsec. If there is more than one sender, the sequence numbers (to avoid replay attacks) have to be synchronised between the senders. Another possibility would be that the receivers have to track one sequence number for each sender which would require a change of the current SA implementations. Another problem is the key distribution. Since IKE uses the DH algorithm, it is not easy to derive a common secret key in a group without big adaptions to existing implementations. A solution would be the use of an explicit key server which would result in too much overhead for small networks. Furthermore, this key server would represent a single point of failure.

The Internet Engineering Task Force (IETF) has formed the ''Multicast Security Working Group'' which focuses on securing multicast traffic with a single source and a large number of recipients. One of the released RFCs specifies the ''Multicast Group Security Architecture'' (cf. [46]). The reference framework is intended to encrypt many-to-many multicast too, but every receiver inside the group has to know the IP addresses of the senders in advance. In [46] it is mentioned, that the framework is intended to secure large groups, whereas it can also be used in smaller ones, but not very efficiently. But since the number of group members and their addresses have to be known in advance, dynamic multicast groups cannot be secured in an efficient manner.

3.1.7 Conclusion

An evaluation of IPsec in [47] concludes, that it is far too complex and this complexity leads to a large number of ambiguities, contradictions, inefficiencies and weaknesses.

In principle, IPsec can be used as security extension in KNXnet/IP networks. According to the requirements stated above, ESP can be used, while AH is not suitable. However, due to the high complexity of the IPsec specification, the demands regarding device resources and the mentioned problems with multicast, IPsec is not advisable for networks containing embedded devices. If more powerful devices are available, IPsec can be used to secure KNXnet/IP tunneling or other KNXnet/IP unicast services. For KNXnet/IP routing, special implementations of IPsec would be required.

3.2 SSL/TLS

Secure Sockets Layer (SSL) and its successor Secure Sockets Layer (TLS) [48] is a protocol developed for securing communication between two parties. The specification can be found under [48]. During the initial handshake, the used cryptographic algorithms are negotiated, secret keys for the secured channel may be exchanged, and the entities can be authenticated. After this initial handshake, a secured channel between the two communication parties is established. Like IPsec, the initial key exchange during the handshake is usually done using asymmetric algorithms. The secured channel uses symmetric algorithms for guaranteeing data integrity, freshness, and confidentiality.

TLS offers similar algorithms to IPsec but operates on a higher level in the OSI layer model (cf. Figure 3.2). It encrypts data between the transport layer (e.g. TCP or UDP) and the application layer. Besides the classical utilisation of TLS for securing Hypertext Transfer Protocol (HTTP) traffic, it can be used to secure all kinds of application protocols. It is even possible to tunnel the entire network stack and set up VPNs using TLS (cf. Section 3.3).

Since TLS does not modify the IP, it is transparent to the network infrastructure and there is no need to adapt routing and switching hardware inside the network. Therefore the integration in existing installations is usually easier compared to the use of IPsec.

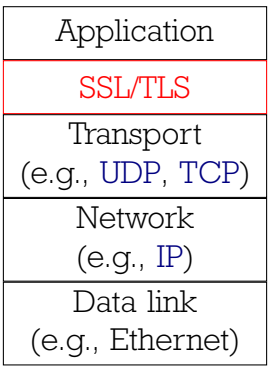


Figure 3.2: Position of TLS in the OSI layer model

3.2.1 Cipher suites

TLS is very flexible with respect to the use of algorithms. So it is possible to implement secured unicast connections on embedded devices, too. [49] shows an implementation of a complete secure web server, using HTTP and TLS. In this implementation, the initial handshake of TLS is done with the help of ECC [23]. The web server runs on very constrained devices like a 4MHz 8-bit ATmega processor. The extension of TLS for elliptic curve cipher suites is specified in [50].

TLS specifies a number of different cipher suites. The suite defines the combination of the key exchange algorithm, the entity authentication mechanism, the hashing function, integrity protection and the encryption algorithm. An example of a cipher suite would be `TLS_RSA_WITH_AES_128_CBC_SHA`, which defines the use of RSA key exchange, AES encryption with 128 bit key length and CBC block cipher mode as well as the use of the SHA hashing function. The suite `TLS_NULL_WITH_NULL_NULL` is the initial cipher suite of any TLS connection and provides no protection.

3.2.2 TLS handshake

The stated requirements for security in HBA systems would result in a "full handshake" that verifies both, the sender and the receiver. Since this full handshake of TLS is used as reference for the presented protocol in Chapter 4, it is explained in more detail. Figure 3.3 and the explanation below describe the handshake messages in case of mutual entity authentication.

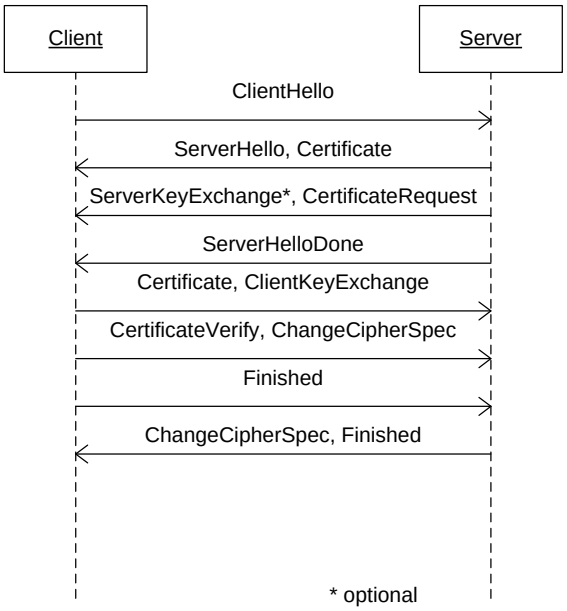


Figure 3.3: full TLS handshake

The client initiates the communication by sending a `ClientHello` message to the server. The fields of this message are shown in Figure 3.4.

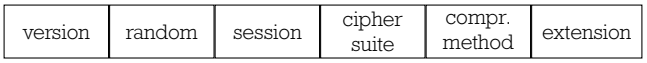


Figure 3.4: TLS Client-/Server Hello message

The first field contains the TLS protocol version, followed by a random value (together with a timestamp) and a session identifier. This session identifier is used to correlate server responses to this request and to resume previous sessions. The next three fields announce the client side available cipher suites, compressions methods and extensions.

The server responds with `ServerHello`, containing also the fields shown in Figure 3.4. After the protocol version, a new random value and the session identifier from the `ClientHello` message is transmitted. The last 3 fields indicate the chosen cipher suite, compression method and extension, out of the lists provided in the `ClientHello` message. The server continues by sending its certificate embedded in the `Certificate` message. Depending on the negotiated key exchange algorithm (which is part of the chosen cipher suite), the server has to provide additional infor-

mation for the key exchange in the ServerKeyExchange message. This message could contain the prime modulus and the generator used for the DH operation, in case of an anonymous DH key exchange. After the server requests the certificate from the client by sending CertificateRequest, ServerHelloDone is sent to the client indicating the end of the hello messages.

Since the server requests a certificate, the client sends its certificate embedded in the Certificate message. The following ClientKeyExchange contains in case of anonymous DH key exchange, the parameters for the calculation of the pre-master secret. These parameters and the one of the ServerKeyExchange are used on both sides via the DH algorithm to calculate a shared secret (cf. Section 2.9.3), which is used as pre-master secret[1] and sends it (encrypted) with the ClientKeyExchange. If DH is not used, the client calculates the pre-master secret message. In contrast to the ServerKeyExchange message, the ClientKeyExchange is mandatory. To prevent replay attacks, the client has to send the CertificateVerify message. It contains the digital signature of the messages sent up to now during the handshake process. This explicitly verifies possession of the private key in the certificate. The message ChangeCipherSpec applies the negotiated cipher. The Finished message is the first one protected with the just negotiated algorithms, guaranteeing that the handshake was successful. It also contains a hash of the handshake messages.

If the server receives a valid Finished message, it changes its cipher specification too by sending the ChangeCipherSpec message and completes the handshake with a Finished message.

Note that the verification of the client certificate is done explicitly by the server by validating the CertificateVerify message, while certificate of the server is verified implicitly by the client during the handshake.

To ensure data freshness, TLS uses a sequence number for the MAC calculation. The number is 64 bits long and must not wrap. If a TLS connection would need to wrap the sequence counter, the connection must be renegotiated.

[1] Both parties use the same algorithm to convert the pre-master secret (which has a variable length – depending on the used key exchange algorithm) to the master secret (which has a fixed length of 48 bytes). Afterwards, the pre-master secret is not used anymore.

3.2.3 Conclusion

TLS offers a fast and well-established handshake protocol for the mutual authentication and protocol negotiation.

Since TLS is a unicast protocol (asymmetric key exchange is done using unicast algorithms – there are no extensions for multiparty key exchange), it can only be used to secure KNXnet/IP unicast services. It is not usable for multicast environments like KNXnet/IP routing.

Nevertheless, because of the lightweight handshake for authentication, TLS served as basis for the developed protocol which will be presented in Chapter 4.

3.3 VPN

A Virtual Private Network (VPN) is a logical network that is built upon a foreign network called host network [51]. The VPN can also use mechanisms to secure the communication in the logical network (called secure VPN). The VPN is transparent to the connected devices. Typically, secure VPNs use other protocols like IPsec or TLS as underlying protocol for the secure communication. The secure VPN software itself handles the connections and provides additional and more comfortable functions. A popular secure VPN implementation is OpenVPN [52].

OpenVPN provides several methods to ensure authentication: a pre-shared symmetric key, a username and password combination, a TLS certificate, or a combination of these methods. While IPsec manipulates the network protocol layer, OpenVPN encapsulates the encrypted VPN packets into untouched TCP or UDP packets of the host network. However, the VPN packets themselves are secured down to IP layer (Layer 3) in routing mode or down to Ethernet (layer 2) in bridging mode.

A centralised server controls the connections and each OpenVPN client connects via a secure point-to-point connection to the server. This means that the server has to keep track of several connections with different keys. The whole traffic within this network is routed through these tunneling connections to the server (cf. Figure 3.5).

This is also true for multicast communication. Multicast messages are sent to the server and distributed there. So the server has to decrypt each multicast message once and

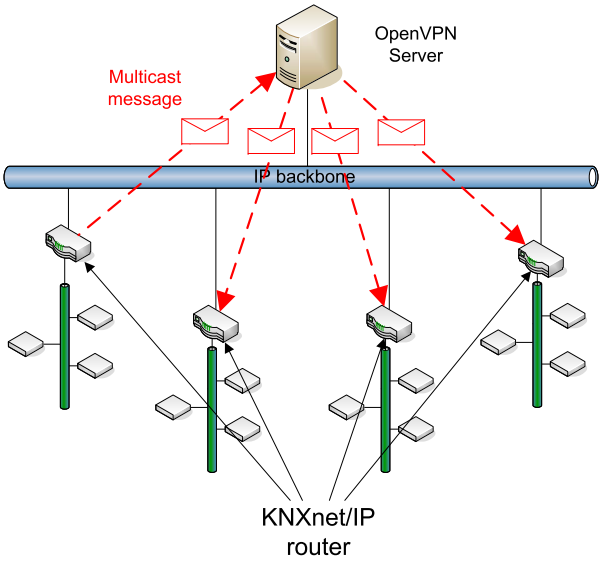


Figure 3.5: OpenVPN multicast

encrypt it again several times for all other parties. This results in a high demand on bandwidth, memory (for storing the different session keys), and computational power at the server. Therefore, the server represents a bottleneck and a single point of failure for the whole network especially if multicast is used like in KNXnet/IP routing. Furthermore, the required resources at the server will exceed the capabilities of an embedded device and so more powerful devices would have to be used as OpenVPN servers. OpenVPN may be used to secure KNXnet/IP tunneling environments and probably small KNXnet/IP routing networks, but in general a OpenVPN solution is of limited use for KNXnet/IP.

3.4 Secure Shell (SSH)

The term SSH is ambiguous. As the name suggests, SSH is a program suite which provides the possibility to log into a remote computer using a secure channel. While other programs and their protocols like telnet transmit the whole traffic unencrypted (including username and password), SSH encrypts the traffic and is therefore usable to connect to a remote service via an insecure network like the Internet. Coming from Unix systems, server and client programs are available for most of the modern operating systems.

Since the underlying protocol for the SSH programs, which is also called SSH, is very flexible, many other applications were built around the SSH protocol. Even the standard clients usually support extended features, like the forwarding of screen output and tunneling like TCP port forwarding. With the help of these mechanisms, single TCP connections can be tunneled via an SSH connection. A more advanced implementation can build up a full featured secure VPN with the help of the SSH protocol. In the OSI reference model, the SSH protocol is situated on the same level like TLS (cf. Figure 3.6).

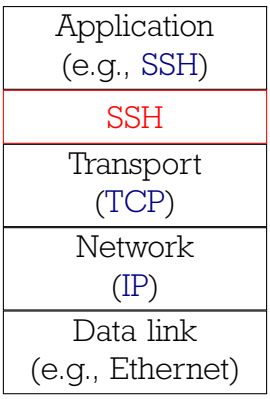


Figure 3.6: Position of the SSH protocol in the OSI layer model

The protocol architecture is specified in [53]. There are three major components, which form the SSH protocol: The Transport Layer Protocol, which ensures server authentication, confidentiality, and integrity. The second part, the User Authentication Protocol authenticates the client to the server. The third part, the Connection Protocol multiplexes the encrypted tunnel into several logical channels. The details to these components can be found in the RFCs [54, 55, 56].

The Transport Layer Protocol (cf. [54]) authenticates the server, exchanges keys, encrypts the traffic and provides integrity protection. Similar to TLS, all of the used algorithms are negotiated in advance by this protocol. The protocol is designed to operate with almost any asymmetric algorithm. But up to now, no ECC key exchange algorithm has been specified within the SSH protocol. The main problem is, that the SSH specification handles only point-to-point connection with a client/server relationship. Single connections with more than two participants are not taken into consideration. Like TLS, SSH is a unicast protocol which uses point-to-point key exchange algorithms. Therefore it is not usable for multicast environments.

3.5 Summary

The analysis of proven IT security mechanisms showed that each one has its advantages and disadvantages, but none of them fully meets the requirements of KNXnet/IP. While IPsec is only of limited use for embedded devices and has problems with multicast, TLS and SSH completely lack multicast support. VPN implementations like OpenVPN have a bad performance in large networks due to conceptual reasons. Table 3.2 shows the results of the comparison of the analysed IT security mechanisms with respect to the stated requirements in Section 1.3.

	Entity auth.	Secured channel	Multicast	Resource consumption
IPsec	+	+	∼[2]	–
SSL/TLS	+	+	–	∼[3]
SSH	+	+	–	∼[3]
OpenVPN	+	+	–[4]	–[4]

Table 3.2: Comparison of security protocols

Since none of the compared protocols meets all stated requirements, Chapter 4 comes up with a new protocol.

[2] See notes in Section 3.1.
[3] If special encryption-mechanisms like ECC are used.
[4] In case of many clients.

Chapter 4

Solution

According to the conclusion of Chapter 3, a new security extension to protect KNXnet/IP communication is presented in this thesis. The basic protocol architecture of this extension is shown in Figure 4.1. The security layer is located between the network layer (UDP or TCP) and the KNXnet/IP layer. The main advantage of this architecture is that a change of the KNXnet/IP frame format is not necessary. Thus, it is not limited to KNXnet/IP and can also be applied to other technologies (cf. [8]).

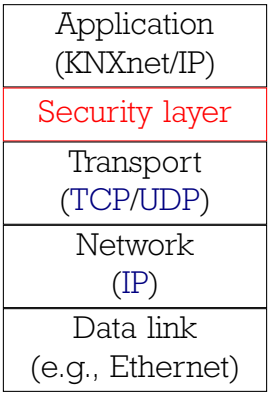


Figure 4.1: Protocol position in the OSI layer model

To be able to securely communicate with other KNXnet/IP devices, a KNXnet/IP device has to establish a secure communication relation to the other communication parties. This communication setup is divided into three phases: *configuration*, *key set distribution*, and *secure communication*. In the configuration phase, the ETS[1] acts as CA (cf. Section 2.9.3) and signs the public keys of the KNXnet/IP devices to create certificates. These certificates together with the public key of the ETS are distributed to the

[1] The ETS is the standardised and only software for the programming and configuration of KNX devices.

KNXnet/IP devices. In the key set distribution phase, the KNXnet/IP devices can authenticate themselves and agree on a common set of keys. This set of keys is used in the third phase for setting up the secured channel, which is used to exchange KNXnet/IP messages during normal operation.

4.1 Configuration using ETS

Figure 4.2[2] shows the operation sequence of the configuration phase. During configuration, each KNXnet/IP device creates its own key pair for asymmetric encryption. While the private key remains secret to the device, the public key is transferred via a physically secured channel to the ETS. The ETS acts as CA. It signs the received public key to create a certificate and transmits the certificate together with the ETS's public key back to the KNXnet/IP device. The certificate consists of the signed public key and the IP address of the KNXnet/IP device. With the help of this certificate, the KNXnet/IP device can authenticate itselves during the key set distribution process.

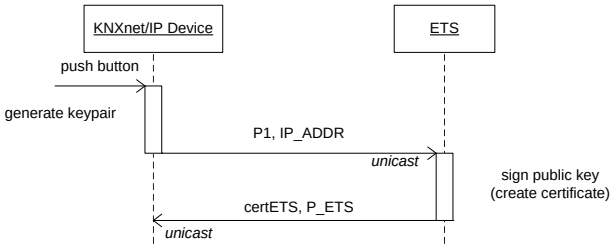


Figure 4.2: Configuration phase

Since the configuration phase is used to set up the initial security token (i.e., the public key of the CA and its certificate), establishing a secured channel using cryptographic techniques is not possible during the configuration phase. Therefore, the configuration process has to be performed in a physically secured environment. One possibility is to directly connect the KNXnet/IP device to the ETS via a point-to-point connection

[2]For the rest of this thesis, the following notation is used: K denotes a private key, P a public key, KS a key set, SK a symmetric key, N a number used only once (called nonce), and $cert$ a certificate. Furthermore, $sign(X, text)$ denotes the signature calculation of $text$ using the secret key X, $encr(X, text)$ the asymmetric encryption of $text$ using the secret key X, $secured(X, text)$ a secured (symmetric encryption) transmission of $text$ using key set X, and || the concatenation operator.

(e.g., using a local EIA-232 interface). This direct connection reduces the untrusted environment to a minimum.

4.2 Key set distribution

After a KNXnet/IP device has been configured accordingly, it is in possession of three security tokens: the public key of the CA (i.e., the public key of the installation's ETS), its own public/private key pair, and its certificate that proves the authenticity of the public key. Using these tokens, a KNXnet/IP device is able to start the second phase where the key sets for the secure communication phase are distributed.

To distribute these key sets, some kind of *key set distribution protocol* is necessary. As mentioned in Section 1.3, KNXnet/IP specifies unicast and multicast services. Therefore, a protocol is required that supports a distribution not only between two devices (e.g., for KNXnet/IP tunneling) but also among a multicast group (e.g., for KNXnet/IP routing). However, since a unicast communication relation can be seen as a special multicast group consisting of two group members, the proposed KNXnet/IP security extension uses a single group key set distribution protocol for both kinds of services.

4.2.1 Unicast

In the proposed key set distribution protocol, the key sets for secure communication are maintained by a coordinator. In the case of unicast, the KNXnet/IP server automatically acts as coordinator.

The key set distribution protocol works as follows: Suppose, for example, a KNXnet/IP client wants to set up a secured channel to a KNXnet/IP server (cf. Figure 4.3). To retrieve the key set from the KNXnet/IP server, the KNXnet/IP client sends a `hello`-message to the KNXnet/IP server. This message is signed with the private key of the client and contains a nonce N_1 as well as the certificate of the client. The KNXnet/IP server receives this message and verifies the certificate, extracts the client's public key and verifies the signature. If the signature is valid, the server immediately responds to the `hello`-message with a signed `coord_avail`-message, containing nonce N_1 of the `hello`-message and a new nonce N_2. N_1 relates the response of the KNXnet/IP

server to the actual `hello`-message and avoids that someone replays the server's response to an older `hello`-message (cf. [57]). To prove the identity of the server, the client verifies the signature, extracts the server's public key and verifies the certificate. If the certificate is valid, the identity of the server has been proven. Afterwards, the client requests the key set from the server by sending a signed `key_req`-message – containing the nonce N_2 which relates this message to the `coord_avail`-message. Since N_2 was chosen by the server, the identity of the client has also been proven (cf. Section 4.4.1) and mutual entity authentication is established. Again, to avoid replay attacks in the next step, a newly created nonce N_3 is included in the message. The coordinator responds with a `key_resp`-message, containing the number N_3 and the key set itself. The message itself is signed with the private key of the coordinator (K_2) and encrypted with the public key of the KNXnet/IP client (P_1).

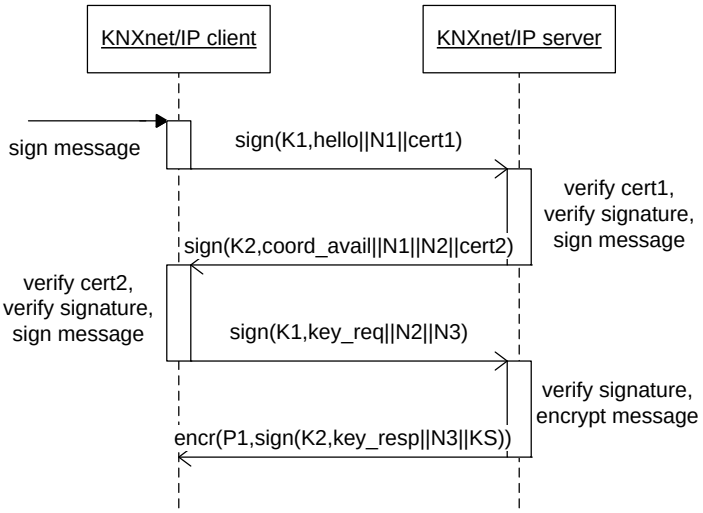


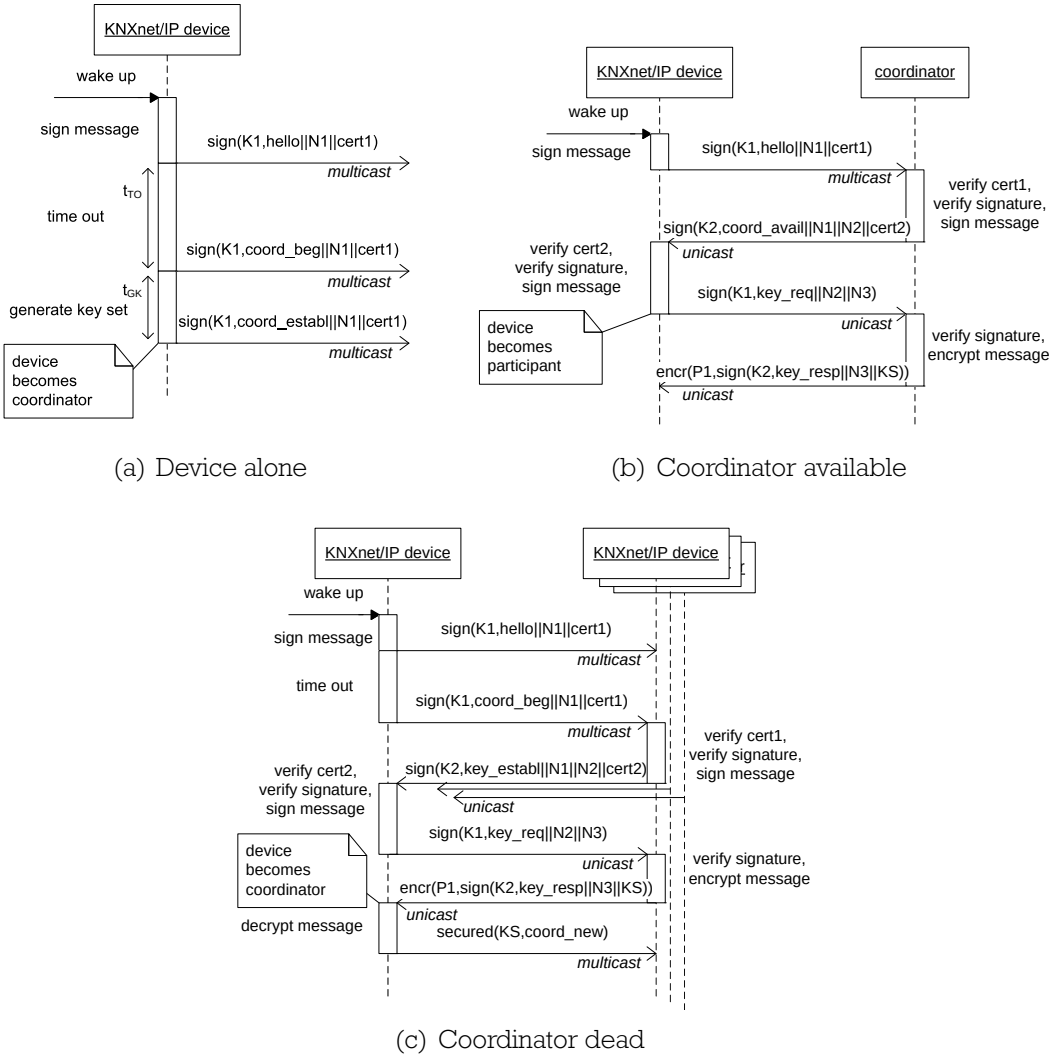
Figure 4.3: Key set distribution for unicast services

4.2.2 Multicast

In the case of multicast services, the role of the coordinator is not predefined. Therefore, the KNXnet/IP devices that are members of the same multicast group[3] must agree on a common coordinator. This works as follows: If a KNXnet/IP device wants to join a

[3]For the rest of this thesis, the term "multicast group" denotes an IP multicast group and not a KNX multicast group.

multicast group (e.g., KNXnet/IP router after power up), it sends a `hello`-message to the multicast group. Again, this message is signed with the private key of the sender and contains a nonce N_1 as well as the certificate. Depending on the state of the multicast group, four different cases are distinguished:



(a) Device alone

(b) Coordinator available

(c) Coordinator dead

Figure 4.4: Coordinator scenarios

1. Figure 4.4(a): If there is no response from other KNXnet/IP devices within the timeout t_{TO}, the sender of the `hello`-message assumes that it is the first active device within the group and so it becomes the coordinator. To announce that it assumes the coordinator role and to avoid that a second device tries to become coordinator at the same time, it will send a `coord_beg`-message. After having sent this announce message, it generates the key set for this group. The gen-

eration process takes the time t_{GK}. When the coordinator has finished the key set generation process, it sends out the message `coord_establ`. Note, that the timeout t_{TO} shall be greater than the key generation time t_{GK} to avoid conflicts with other devices that try to become coordinator at the same time (cf. Case 4).

2. Figure 4.4(b): If there is a coordinator available in the group, it immediately responds to the `hello`-message with a signed `coord_avail`-message, containing the nonce N_1 of the `hello`-message and a new nonce N_2. Again, N_1 relates the response of the coordinator to the actual `hello`-message and avoids that someone replays the coordinator's response to an older `hello`-message. After having received the response from the coordinator, the sender of the `hello`-message is able to request the group key set from the coordinator by sending a signed `key_req`-message. This message contains the nonce N_2 which relates this message to the `coord_avail`-message. Again, to avoid replay attacks in the next step, a newly created nonce N_3 is included in the message. The coordinator responds with the encrypted `key_resp`-message, containing N_3 and the key set itself.

3. Figure 4.4(c): If there is a coordinator but it does not respond for any reason (e.g. the coordinator crashed), the new device takes over the role of the group's coordinator by sending a `coord_begin`-message (cf. Case 1). If there are other group members, they immediately respond with a signed `key_establ`-message, informing the new coordinator that there is already a group key set available. This message contains the nonce N_1 of the `coord_begin`-message and a new nonce N_2. The new device chooses one of the responding group members for authentication and obtains the key set from it by sending a `key_req`-message. After having received the key set, the new device takes over the coordinator role by sending a `coord_new`-message. This message is encrypted (symmetric) with the key set.
   When there are more than one devices entering this described scenario at the same time, only one of them will be the new coordinator, since this conflict is resolved like described in Case 4.

4. This section describes the behavior of the protocol in case of a conflict. The conflicting scenarios can occur, if during the setup of the coordinator (cf. Figure 4.4(a)), more than 1 device is competing for the coordinator role. The devices are distinguished via their IP-addresses. Extrema (min or max) defines the

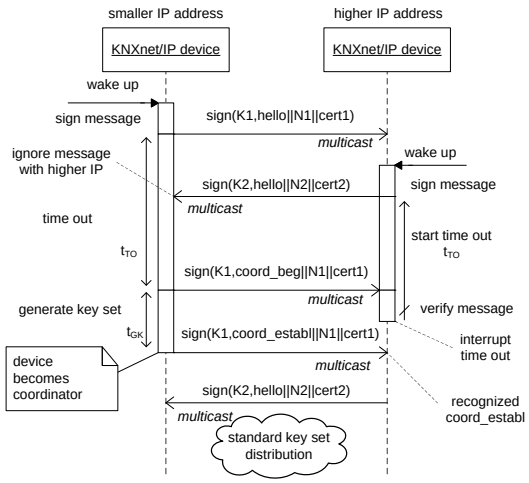
winner of a possible competition. The conflicting scenarios are split up into three parts. In the examples it is assumed, that the device with the smaller IP address is the privileged one and wins the competition.

a) In Figure 4.5(a) the device with the smaller IP address is the first one which sends a message. According to the standard case shown in Figure 4.4(a), it starts with the `hello` message, waiting for a response from a possibly available coordinator. A `hello` message from a device with a higher IP address will be ignored during this waiting period. The ''winning'' device with the smaller address continues like in the standard case by sending the `coord_beg` message. When the device with the higher address receives and validates this message, it can stop the timer and waits for the finishing `coord_establ` message. After this message, it can begin requesting the common key set from the new coordinator.

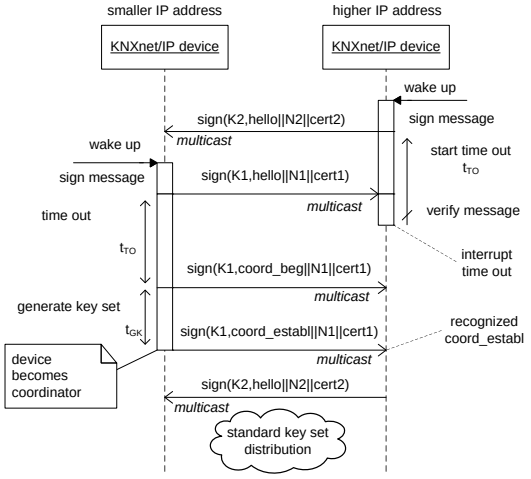
b) In the next case shown in Figure 4.5(b) the device with the higher IP address is already alive, when the device with the smaller address wakes up. Since the new device did not recognise the initial `hello` message, it starts by sending out its initial message. Because of the smaller IP address of the new message, the device with the higher address will abort its starting sequence silently. After the second device established its coordinator position by sending `coord_establ`, every other device can start communicating with the new coordinator.

c) In scenario of Figure 4.5(c) the second device wakes up during the key set generation of the first device. The `hello` message of the new device will be ignored by the first device. The end of the key set generation after time t_{GK}, which is denoted by the `coord_establ` message, happens before the end of the time out running at the second device (time t_{TO}). This is guaranteed by the inequation $t_{GK} < t_{TO}$ (message transmission times and a safety margin should be added to t_{GK}). By the reception of the `coord_establ` message, the second device aborts its timer and restarts by sending the `hello` message.

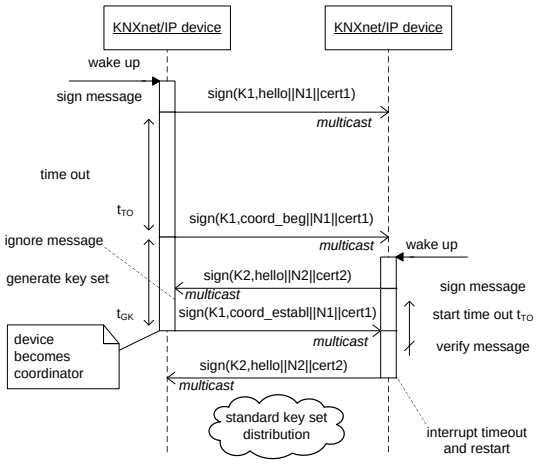
When the coordinator is established, each KNXnet/IP device can send a `key_req`-message to receive the group key set in the `key_resp`-message from the coordinator.



(a) Conflict 1: privileged device ignores message



(b) Conflict 2: unprivileged device is interrupted



(c) Conflict 3: device ignores message after `coord_beg`

Figure 4.5: Conflicting coordinator scenarios

4.2.3 Blacklist

Besides the security objectives mentioned in Section 2.2, which can be achieved by the means of cryptography, to guarantee the availability of a service is another topic when talking about security. Common attacks against this provision are the Denial-of-Service (DoS) attacks.

In contrast to other security threats, the attacker does not want to gain access to a system or secret data, but wants to disturb the normal operation. In case of a Web server, the attacker will put too much load on the server that either the server's resources are exceeded or the communication line to the server is overloaded. In both cases the Web server will not be able to handle normal requests – the attacker succeeded.

Due to its ad-hoc nature, it is not possible to completely protect a system from DoS attacks. But some organisational countermeasures can be taken to make these attacks more difficult. A common approach to make DoS attacks more difficult during authentication is the use of client puzzles (cf. [58]). The server asks the client to solve a computational intensive riddle which the client has to solve before it can continue with the authentication process. This puts load on the client and the attacker has to solve a lot of riddles at the same time to succeed. Since we are dealing with embedded devices with limited resources and an attacker may use a much more powerful machine to start the attack, this kind of approaches is of limited use here.

In our scenario, an attacker can replay valid key exchange messages to put a lot of load on the coordinator. As an effective method against this DoS attack, a common blacklist is maintained. This blacklist records the key information of replayed protocol messages. This approach does not prevent an exhaustion of network bandwith, but it is able to reduce the load on the server since it can simply ignore blacklisted messages rather than performing intensive cryptographic operations.

If an attacker tries to gain access to the system or starts a DoS attack by replaying one or more messages from the address IP_a containing some nonce N_a, this combination of address and nonce will be blacklisted when it is recognised that is has been used by an attacker (cf. Section 4.4.1). To save computational power and to limit the working surface for possible DoS attacks, messages from the blacklisted address IP_a containing the nonce N_a will be ignored without any treatment (with one exception).

Since it is possible, that a device chooses a random number which has been used previously by an attacker (e.g. after a power-up when the blacklisted nonces are cached in volatile memory), this device would establish a new coordinator position because there is no response from other devices. To avoid a second ''valid'' coordinator, this device has to be informed by other devices, that it is using a blacklisted address/random number combination. This is done by sending a `blacklisted` message (cf. Figure 4.6). The wrongfully ignored device has to choose another random number and restart the protocol.

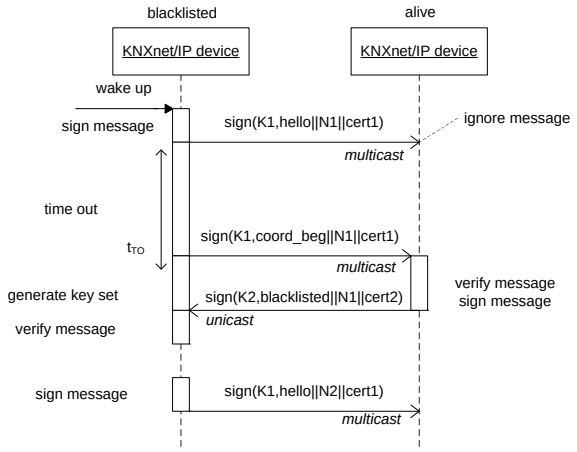


Figure 4.6: Inform blacklisted device

To avoid the revocation of all address/nonce combinations in a long term environment (which would result in a DoS), the blacklisted entries have an expiration date. When this date is reached, this entry will be removed from the blacklist and the banished combination can be used again.

4.2.4 Certificate revocation

Since it is possible that a KNXnet/IP device gets compromised, a revocation list is required that contains the certificates of all compromised KNXnet/IP devices. This revocation list is maintained by the ETS and has to be distributed to all KNXnet/IP devices when a new certificate is added to the list. This is done by sending a `cert_revoke`-message that is signed with the private key of the ETS. Since all KNXnet/IP devices are

in possession of the public key of the ETS, malicious `cert_revoke`-messages can be identified by verifying the signature.

4.3 Secure communication

After a key set has been distributed during the second phase, the secured channel between the communication parties can be established. To guarantee the security objectives of the secured channel, symmetric algorithms are used, exclusively. Besides the advantage of higher processing speed, all multicast parties have to share only a single key. In contrast to asymmetric algorithms, a message has to be processed only once since all KNXnet/IP devices of a communication relation (e.g., a single multicast group) use the same key set (cf. Section 2.4.1).

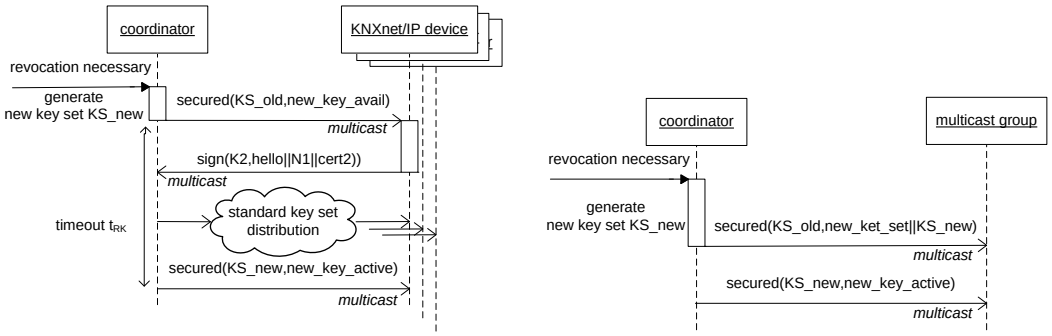
A key set consists of 3 security tokens: a secret key SK_C for en- and decryption, a secret key SK_{MAC} for the MAC calculation, and the current valid sequence counter C. The used frame format (cf. Figure 5.11) is based on the application protocol frame format of TLS 1.2 (cf. Section 3.2).

Confidentiality is guaranteed by using AES-128. To ensure data integrity and data freshness, HMAC (cf. Section 2.5.1) in combination with SHA-256 (cf. Section 2.5.1) and a sequence number are used. As already described in Section 2.5.1, HMAC provides an integrity check based on a secret key. This key must be available on all devices to create valid hash values and to validate the messages.

The sequence number ensures data freshness. Besides the sequence counter, also the IV could be used to check the actuality of a datagram. But in contrast to the strictly increasing sequence counter, the IV is random. Therefore every used IV value has to be stored to identify replayed messages. When using a sequence counter, only messages with a higher counter value are accepted. Since the underlying protocol possibly does not maintain the original message order, the devices may store only a limited amount of missing sequence numbers to identify ''old'', but not replayed messages.

4.3.1 Key set revocation

Section 2.9 describes two situations, where rekeying is necessary. In these situations, the coordinator has to create a new key set. To inform the other communication parties that a new key set is available, a `new_key_avail`-message is sent by the coordinator. In case of a compromised device (cf. Figure 4.7(a)), every device of the multicast group has to request the new key set from the coordinator as shown in Figure 4.4(b). Figure 4.7(b), on the other hand, shows the second case where a key set is renewed to limit the amount of key set uses. In this case, the new key set can be encrypted with the old one and transmitted over the already established secured channel. After a configurable timeout t_{RK}, the coordinator initiates the switch over to the new key set by sending the `new_key_active`-message, making the old key set obsolete.



(a) Revocation in case of a compromised device

(b) Revocation in case of out-dated key set

Figure 4.7: Revocation scenarios

4.4 Security analysis

In this section, the security of the key distribution protocol will be analysed. It is assumed, that an attacker is not able to compromise a device and steal a private key. If the attacker knows a certificate with its private key, he can simulate the behaviour of an authorised device and gain access to the secured multicast group[4].

[4]Nevertheless, the stolen certificate can be added to the revocation list if the attacker is identified as described in Section 4.2.4.

It is obvious, that the establishment of the coordinator in Figure 4.4(a) does not offer working surface of attacks (since the device is alone and there are no interfering messages).

The security of the standard key distribution procedure depicted in Figure 4.4(b) is similar to the one of TLS. This protocol contains the same security primitives as the full TLS handshake without DH key exchange (Figure 3.3). The $\texttt{ClientHello}_T$[5] message corresponds to $\texttt{hello}_O$, which additionally carries the certificate of the client. $\texttt{ServerHello}_T$ and $\texttt{Certificate}_T$ are combined in the $\texttt{coord\_avail}_O$ message: the server sends it's certificate and signs the random number of the request message which implicit authenticates the server. The other messages $\texttt{CertificateRequest}_T$ and $\texttt{ServerHelloDone}_T$ do not contain security relevant information and are therefore not necessary in the presented protocol. In TLS, the client answers by sending the $\texttt{Certificate}_T$ message, containing it's certificate. Since the certificate has already been sent in the $\texttt{hello}_O$ message, in the presented protocol the certificate exchange is not necessary any more. $\texttt{CertificateVerify}_T$ includes a digitally signed hash of all previous messages including the received nonces from the server, to verify the presented certificate. In the developed protocol, this is done by $\texttt{key\_req}_O$, containing the digitally signed nonce of the server. Since in TLS the client generates the (premaster) key, it has to send the encrypted key with the $\texttt{ClientKeyExchange}_T$ message to the server. In the presented protocol, the server knows the common key and sends the encrypted key to the client using the $\texttt{key\_resp}_O$ message. Again, the other messages $\texttt{ChangeCipherSpec}_T$ and $\texttt{Finished}_T$ do not contain security relevant information.

The situation is more difficult in case of conflicts during the setup phase of the key distribution protocol, i.e. when establishing a coordinator. The security of the chosen protocol is based on the fact that an attacker cannot create a valid signature of an arbitrary, but defined message. Therefore nonces are part of every message to ensure freshness. Possible replay attacks are discussed in the following section.

[5] Messages written as $\texttt{message}_O$ are part of the protocol presented in this chapter, while messages written as $\texttt{message}_T$ are part of the TLS protocol

4.4.1 Replay attacks

The following analyses refer to the scenarios described in Section 4.2. Every time an attacker is exposed, its address and the used random number will be blacklisted to avoid further DoS attacks. The sniffed address/nonce combination becomes useless (cf. Section 4.2.3).

1. Figure 4.4(a): Since only one device is present (which might be the attacker), this scenario presents no hazards. The attacker could ''establish'' its coordinator position, but it is not able to send valid answers to new requests and will ''loose'' its coordinator position if any other device wakes up (see Case 2).
2. If the attacker simulates the KNXnet/IP device of Figure 4.4(b), he will be able to replay a former `hello` message. The coordinator will accept this message and respond by sending the `coord_avail` message. Since the attacker is not able to calculate a valid signature for the `key_req` message, the communication will be interrupted here.

   If the attacker pretends to be the coordinator, he will not be able to send a valid `coord_avail` message and the KNXnet/IP device is able to unmask the attacker.
3. Figure 4.4(c): In case of a dead coordinator, the attacker can probably send the previously sniffed `hello` and `coord_beg` messages. Since the `key_req` message must contain the received nonce N_2, again the attacker cannot provide a valid message.

   If the attacker tries to simulate a responding KNXnet/IP device, he will fail by sending a valid `key_establ` message, containing the received nonce N_1.
4. Figure 4.5(a): While the `hello`, `coord_begin` and `coord_establ` messages can be repeated by an attacker, as described before, the standard key set distribution will not work (cf. Case 2). Since the attacker will be blacklisted now, this interruption of the coordinator establishment can only happen once.

   If the attacker fakes the device with the higher IP address, no attack is possible since these messages are ignored. The replication of the last `hello` message is handled in Case 2.
5. Figure 4.5(b): Again, like in the previous case, an attacker imitating the KNXnet/IP device with a smaller address will be exposed during the standard key distribution. But in this case, the attacker is able to stop the execution of other devices. To

avoid this DoS attack, the address and the used random number of the attacker will be blacklisted. This prevents a new interruption of the protocol since the messages from a blacklisted source are ignored.

Due to the fact that the device with the higher address does not interfere the other device, there is no chance for the invader to get access to the common key set.

6. For the scenario of Figure 4.5(c), the same arguments of the previous Case 5 apply.

Chapter 5

Implementation

The prototype implementation demonstrates the functionality of the presented solution and acts proband for performance tests. This chapter is divided into two main parts: the explanation of the implementation and the test results. Inside Section 5.1, the used test configuration is presented and all participating hard- and software parts are shown. Section 5.2 compares different hardware platforms, with respect to the implementation of ECC algorithms.

5.1 Prototype implementation

During the development of the security extension, a prototype network containing an IP backbone and two connected field networks has been implemented to demonstrate the feasibility of the protocol. This section describes the basic setup of the test configuration and the software architecture of the proof-of-concept implementation. The sub-modules and used libraries are explained and the message formats for the transport of information via the network is presented.

5.1.1 Test configuration

In the testing environment, a prototype KNX network has been set up (cf. Figure 5.1). This network consists of two small KNX TP 1 network segments which are interconnected by an IP backbone. Two AT91SAM7X-EK evaluation boards act as KNXnet/IP

routers. While the on-board Ethernet controller provides the interface to the IP backbone, a TP-UART [59] is used to access the KNX TP 1 network segment. For demonstration purposes, a standard KNX light switch as well as a standard KNX light actuator are connected to the TP 1 networks.

Connected to the hub, a PC running in promiscuous mode is sniffing all packets in the IP network. With the help of Wireshark [60] and the KNXnet/IP dissector [61], IP traffic can be displayed and analysed. Without the security layer presented in this thesis, it is possible to intercept the clear text version of the KNXnet/IP frames. Furthermore, it is possible to replay intercepted KNXnet/IP frames to fool the KNX installation. When the security layer is enabled, the KNXnet/IP frames are secured and cannot be analysed on the PC anymore. Even replay attacks are no longer possible.

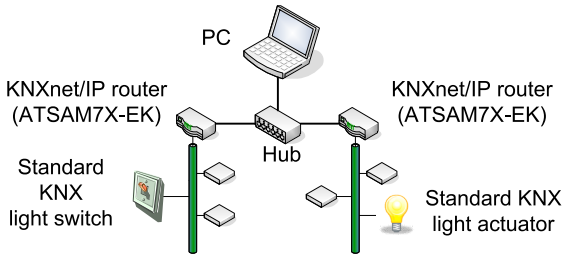


Figure 5.1: Prototype network

5.1.2 Router architecture

In Figure 5.2, the basic hardware layout as well as the software architecture of the secure KNXnet/IP router is shown. The boards are equipped with an AT91SAM7X256 chip which are 32-bit RISC Processors (ARM7TDMI) including 256 KB internal flash memory and 64 KB internal single-cycle access SRAM. In the used evaluation board, the CPU operates at 18 MHz. Via the Media Independent Interface (MII), the CPU is connected to a Davicom DM9161AE Ethernet chip. An ADM3202ARN chip drives the EIA-232 interfaces of the board which are used for the communication with the TP-UART and for debugging output.

The security layer of the software implementation is located between the UDP/IP stack and the KNXnet/IP stack. It is transparent to the KNXnet/IP stack and the layers above.

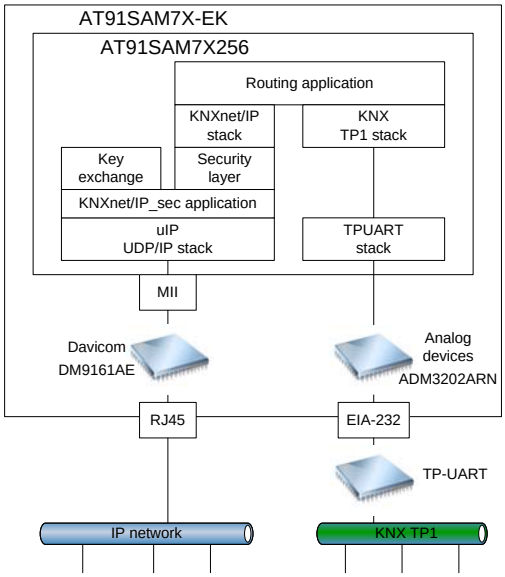


Figure 5.2: Hardware and software overview

The following sections describe the software parts of the proof-of-concept implementation.

uIP UDP/TCP stack

The open source library `uIP` serves as UDP/IP stack. It has been developed at the Swedish Institute of Computer Science [62]. `uIP` features very small code size and low memory usage, hence it is suitable for the use in embedded microcontrollers.

The UDP/TCP stack `uIP` for embedded devices can either be used within a Real Time Operating System (RTOS) or as standalone application. The RTOS FreeRTOS comes with several examples using `uIP` as TCP stack. Probably because of memory issues, other libraries used in the implementation did not work together with `uIP` and FreeRTOS.

Therefore, `uIP` is used as standalone application, which controls the execution of the implemented functions. From `uIP`'s point of view, the whole proof-of-concept implementation (key management, secure communication, KNXnet/IP stack, ...) is a single application. This application (called "`KNXnetIP_sec`") is called when new data was

received by the uIP stack and if a timer event occurred. The timer events are necessary to handle timeouts in the key distribution state machine.

UDP multicast is not supported by uIP natively. Therefore, a patch was written to receive and send multicast packages. Appendix C shows the patches for the uIP code itself (Listings C.4 and C.5) as well as for the ARP part of uIP (Listing C.6).

KNXnet/IP_sec application

The uIP application "KNXnet/IP_sec" receives messages from the UDP stack and forwards it to either the key distribution algorithm or to the security layer. As already mentioned, also timer events are handled and passed to the appropriate sub-application.

Security layer

For cryptographic calculations, many arithmetic operations on large numbers have to be performed. Since native C cannot handle such large numbers, a special library has to be used. The Multiprecision Integer and Rational Arithmetic C Library (MIRACL) supports all of the required functions. Furthermore, it provides functions and examples for many different cryptographic algorithms like AES, RSA, DH, ECC over binary and prime fields, and hashing functions. In addition, MIRACL implements a cryptographically secure PRNG. Optimisations for different processor architectures are written in inline assembly for time-critical routines to enhance speed and exploit special features of some processors. The library also offers some parameters to optimise the memory consumption. The MIRACL library is developed by Shamus Software Ltd. (see [63]).

The prototype implementation uses only static variables instead of heap allocated, dynamic variables. This increases the performance of the code. Since the dimensions of the used variables are known in advance, there is no drawback compared to the dynamic variable allocation.

The security layer uses MIRACL to secure messages. Messages coming from the "KNXnet/IP_sec" application are decrypted and validated and passed to the KNXnet/IP stack. Messages travelling the other way round are encrypted and a MAC or a digital signature is added, before they are sent to the "KNXnet/IP_sec" application.

The symmetric operations for encryption and decryption as well as for MAC generation and verification use the keys which have been agreed by the key distribution module.

The signatures for the key distribution algorithm are calculated and verified with the ECDSA. The used key length is 160 bits. For the encryption and decryption of messages during the key distribution, ECIES is used. Within ECIES, ECDH derives the common secret. As KDF, SHA-256 is used. The symmetric encryption operation for ECIES is AES, HMAC-SHA ensures the integrity of ECIES (both with a key size of 128 bits). During the secure communication, AES-128 ensures confidentiality, HMAC-SHA is used for integrity protection.

Key distribution state machine

Figure 5.3 shows the state machine that has been implemented to realise the proposed key distribution. When a state which is labeled with a `message`, this `message` is sent (e.g. the device wakes up and enters the `hello` state, the `hello` message is sent immediately when entering this state). The labels of the edges of the state machine indicate the conditions for the transition from one state to another (like the reception of a valid `message` or the occurrence of a time out).

The states ''coordinator'' and ''participant'' are accept states, which cannot be left. The device in ''coordinator'' state handles `hello` messages and key requests as described in Section 4.2. The ones in ''participant'' state may have to answer `coord_beg` messages and key requests (when the coordinator is dead).

KNXnet/IP stack

The implemented KNXnet/IP stack does not support all features of the KNXnet/IP specification. Currently, the stack supports the KNXnet/IP Routing service. Extended functionality like lost message handling and other services like KNXnet/IP Tunneling are not implemented. Since the implementation and operation is completely independent from the security layer, the task of the security layer is unaffected by the reduced range of functions of the layers above.

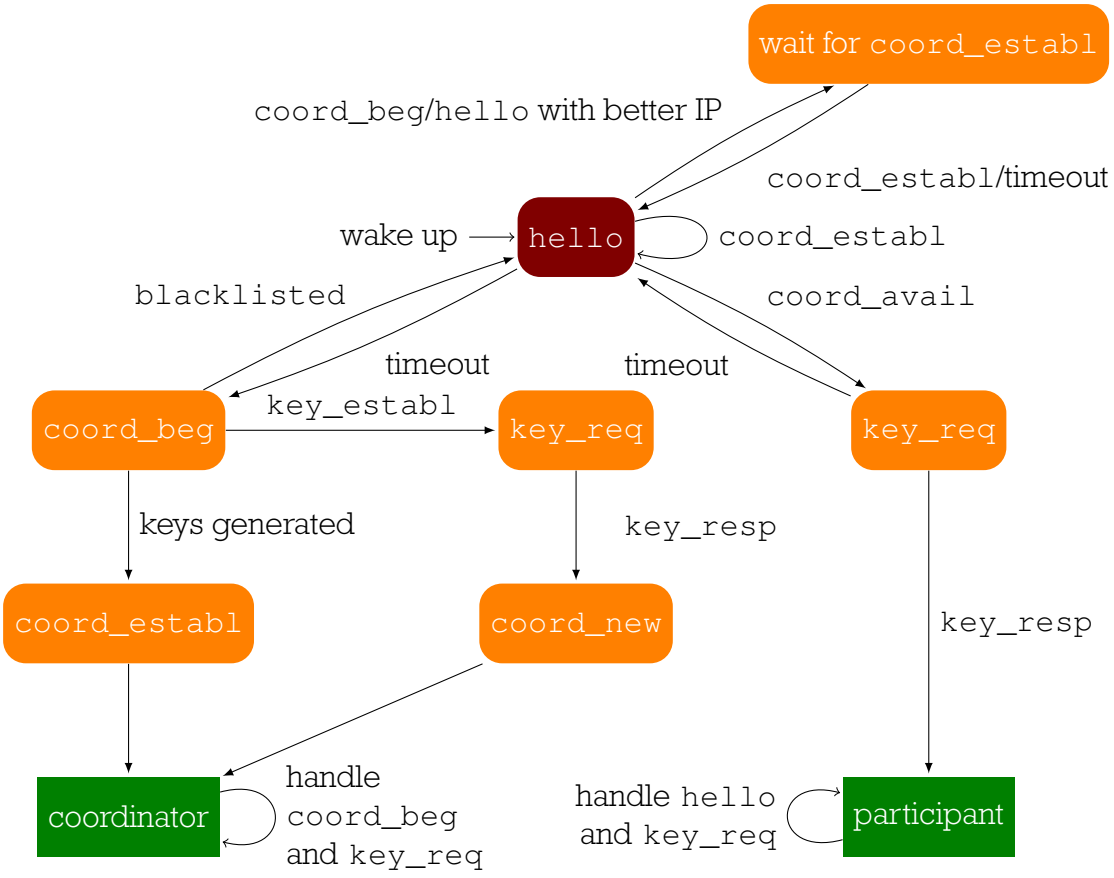


Figure 5.3: Key distribution main state machine

When receiving a message from the routing application, the task of this layer is to prepend the KNXnet/IP header to the received data (cf. Figure 5.4), before it is forwarded to the security layer. Messages from the security layer must start with the KNXnet/IP header, which will be cut off before the message is passed to the routing application. For more information on the KNXnet/IP frame format, see [3].

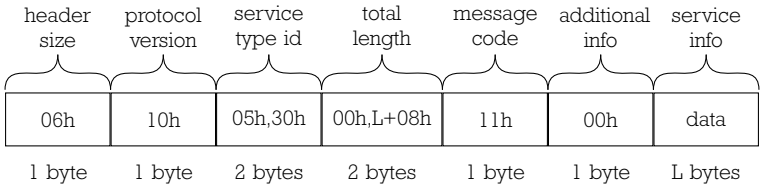


Figure 5.4: Routing indication frame format

Routing application

The routing application sends the messages from the KNXnet/IP stack to the KNX TP 1 stack and vice versa without any further processing. Currently, two interfaces are supported: one TP interface and one interface to the IP network. The routing application can easily be extended to support more than 2 interfaces (requires the implementation of a routing table).

KNX TP1 and TPUART stack

To communicate with KNX TP 1 devices, a KNX TP 1 stack as well as a TP-UART stack are used. The used stacks have been developed at the Vienna University of Technology (cf. [64]).

Other software parts

The own certificate and the public key of the CA have to be available to the KNXnet/IP devices at every time (if the configuration phase is already passed). So it is important that these values reside in memory even if the device is powered off. To archive this, the nonvolatile values have to be written into the flash memory of the board. Listing C.7 demonstrates this process.

Another topic is the seed for the PRNG (cf. Section 2.8). For this proof-of-concept implementation, the Analog/Digital Converter (ADC) is used to get some sort of random values. Since the ADC pins are floating, unpredictable values are read. Nevertheless, since the read values do not vary in a big interval, this random source may not be good enough for productive use. Last but not least, an attacker could put the input pins to a defined level (e.g. ground) to force predictable input values of the ADC.

5.1.3 Security token representation

This section describes the format of the variables and values, which have to be serialised. Field names written in *italic* and without length definition, refer to a previously defined representation.

Public key

The representation of a public key (cf. Figure 5.5) consists of the length field indicating the length of the following public key value. The Least Significant Bit (LSB) of the compressed elliptic curve point (elliptic curve point compression – cf. Section 2.4.2) is the LSB of the length field.

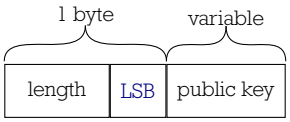


Figure 5.5: Public key representation

Signature

A digital signature consists of the tupel (r, s) (cf. Section 2.5.2). Similar to the representation of the value of the public key, the format of the signature shown in Figure 5.6 has a leading length byte for each following value field.

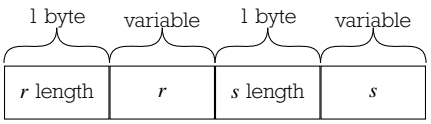


Figure 5.6: Digital signature representation

Certificate

As already described in Section 4.1, a certificate consists of the public key of the KNXnet/IP device, its IP address and the digital signature calculated by the CA. Figure 5.7 shows the format of the certificate representation with the use of the already defined streams of the public key and the signature. The signature is calculated over the public key and the IP address of the device.

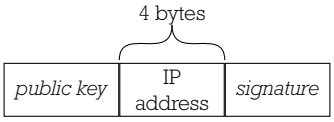


Figure 5.7: Certificate representation

Key set

The key set consists of the AES key and the MAC key. In the serialised representation, the two fixed length fields are simply concatenated (cf. Figure 5.8).

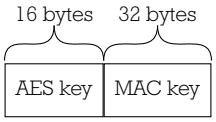


Figure 5.8: Key set representation

5.1.4 Message representation

This section describes the format of the messages, which are sent via the network. Already defined representations of security tokens (cf. Section 5.1.3) are written in *italic*.

Message envelope

All messages which are sent via the network are encapsulated in the structure shown in Figure 5.9. The field "service type" indicates the kind of message. Currently, the following service types shown in Table 5.1.4 are implemented.

Service type	description
KEYEXCHG	Message for the key distribution algorithm
DATA	Data message (KNXnet/IP message)
KEYEXCHG_SYM	Symmetric encrypted key distribution message

Table 5.1: Available service types

If the service type KEYEXCHG is set, the whole key distribution message of Figure 5.10 is used as field "*message*" of Figure 5.9. The service type DATA indicates an encapsulated secured KNXnet/IP frame – a KNXnet/IP frame is used as payload in the "user data" field of Figure 5.11. Messages with KEYEXCHG_SYM service type transport secured frames (Figure 5.11), with a key distribution message (Figure 5.10) as payload.

The integrity of the field "service type" is always protected by the embedded message. E.g. when sending a data message, it is included in the MAC calculation.

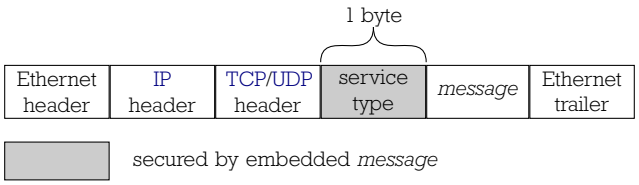


Figure 5.9: Message representation

Key distribution message

Figure 5.10 shows the message format for the key distribution algorithm. The "service type" is the one of the message envelope. It is included in the calculation of the MAC or the digital signature for integrity protection. The "message type" is an integer presentation of the message type like `hello` or `coord_establ` (cf. Table 5.2). The "control field" indicates the presence of the following fields. If the first bit is set, the nonce "N_1"

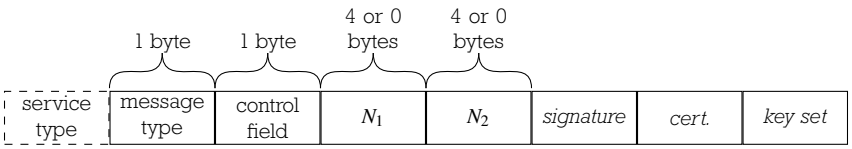


Figure 5.10: Key distribution message representation

is present and is 4 bytes long. There is one bit for each of the following fields N_2, the signature, the certificate and the key set.

Table 5.2 lists the implemented key distribution message types, according to the presented key distribution algorithm (cf. Section 4.2).

Message type	delivery[1]	integr. prot.[2]	encryption[3]	contents
`hello`	m	sign.	no	N_1, certificate
`coord_beg`	m	sign.	no	N_1, certificate
`coord_establ`	m	sign.	no	N_1, certificate
`coord_avail`	u	sign.	no	N_1, N_2, certificate
`coord_new`	m	MAC	sym	no contents
`key_req`	u	sign.	no	N_1, N_2
`key_resp`	u	sign.	asym	N_1, key set
`key_establ`	u	sign.	asym	N_1, N_2, certificate
`blacklisted`	u	sign.	no	N_1, certificate
`new_key_avail`	m	MAC	sym	no contents
`new_key_active`	m	MAC	sym	no contents
`new_key_set`	m	MAC	sym	key set

Table 5.3: Key distribution message types

[1]Message delivery (m = multicast, u = unicast).
[2]Data integrity is guaranteed either via MAC or via digital signature.
[3]Message encryption (sym=symmetric, asym=asymmetric, no=no encryption).

Secured message

As shown in Figure 5.11, a secured frame consists of two different parts. The leading fields are only secured' using HMAC, while the rest of the message is encrypted using AES. Again, the ''service type'' is passed from the message envelope to ensure it's integrity. The first field ''Length'' holds the total length of the message (i.e., sum of ''IV'', ''User data'', ''MAC'', ''Sequence counter'', ''Padding'', and ''Padding length''). The field ''User data'' contains the payload, which can be the KNXnet/IP frame or some key distribution message. The MAC field, which provides data integrity and freshness, is calculated via the following equation

$$MAC = HMAC(SK_{MAC}, Length||IV||User\ data||C||Padding||Padding\ length||IP\ address)$$

where C denotes the current sequence counter. Like in the TLS implementation, the sequence number must not wrap. In the proposed solution, the sequence counter is 32 bit long. The sequence counter is incremented, when the router receives a message from the TP1 network, where data is transmitted with a rate of 9,6 kbit/s. The minimum KNX data frame length on the TP1 network is 8 octets long (containing 1 octet of application data). Between two messages there has to be an idle time of 50 bit times[4]. This means that about 84 messages per second can be received on the TP1 interface. So after about 589 days the sequence counter will overflow, if the worst case assumptions hold (which in fact is very unrealistic). With more realistic assumptions like 2 messages per second, the sequence counter will last for more than 68 years. Note that even in the presented worst case scenario the value domain of the sequence counter is big enough, since in case of rekeying (which is performed periodically – cf. Section 4.3.1) the counter is reset automatically. In the implementation a device has to track one sequence counter for each sender. The senders are identified by the IP address. Hence, a single sequence counter value is only valid once for every address. To ''bind'' a message to a specific sender address, the senders address itself must be part of the calculation of the HMAC tag. Otherwise the same message could be replayed from another IP address without being noticed.

To guarantee data confidentiality, AES-128 in CBC mode is used (cf. Section 2.4.1).

[4]These values are the absolute lower bound for TP1 message lengths and are therefore very unrealistic. Usually, every data frame is acknowledged by the receiver, resulting in another 15 bit idle time on the bus. Besides, it is very unlikely that two consecutive messages contain only 1 octet of application data. Nevertheless, this is a very tough lower bound for the worst case estimation.

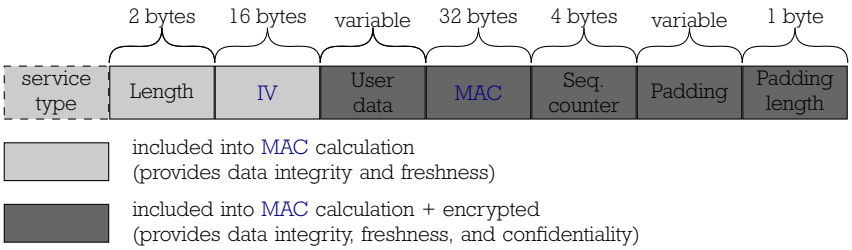


Figure 5.11: Secured frame format

The main advantages of AES are best performance and patent free implementation. The encryption process works as follows:

$$CIPHERTEXT = AES\_128\_CBC(SK_C, IV, User\_data||MAC||C||Padding||Padding\ length)$$

The AES_128_CBC encryption process has three parameters as input: the shared secret key SK_C, the IV, and the plain text (i.e., $User\_data||MAC||C||Padding||Padding\ length$). The usage of the IV within the en- and decryption operation is described in Section 2.4.1. To avoid an attack on CBC mode (cf. [65]), a random number has to be chosen as IV, every time an encryption takes place. Since a block cipher operates only on fixed sized message blocks, the message may be padded up to the necessary block length (cf. Section 2.4.1). This is done by appending "padding" and "padding length" to the message. The code which builds this frame and does all the calculations and a description of this piece of code can be found in Appendix C, Listing C.3.

5.2 Benchmark

Table 5.4 compares the execution speed of ECC operations in a prime and a binary field. Generation refers to the generation of the private/public key pair, each of 160 bits. Signature is the time for calculating a signature (inclusive the needed hash operation) of a 14 characters long message. Verification is the time needed for the successful verification of the previously calculated signature. The column ECIES encryption denotes the encryption using the ECIES encryption scheme of an about 50 characters long message. This time includes the generation of the one-time-key pair. The last column shows the calculation time for the decryption operation.

Operation	prime field	binary field
Generation	470 ms	248 ms
Signature	250 ms	136 ms
Verification	340 ms	240 ms
ECIES encryption	764 ms	384 ms
ECIES decryption	285 ms	135 ms

Table 5.5: Benchmark results

The message lengths are mentioned for completeness' sake. The treatment of longer or smaller messages did not influence the values significantly.

Chapter 6

Conclusion and Outlook

Security can no longer be neglected in today's home and building automation systems. This is especially true for HBA systems where IP is used as medium for backbone networks. This thesis starts with a brief introduction to IP backbones and their requirements regarding a secure communication. The focus is on the popular standard KNX and its extension KNXnet/IP since neither KNX nor KNXnet/IP support reasonable security mechanisms. In Chapter 2, a basic introduction to the field of cryptography is given. The described algorithms are categorised by the security objectives and the underlying mathematical problems. In the following Chapter 3, already available and well-established security mechanisms are analysed. In principle, all of the examined protocols can be used to secure IP traffic. However, it turned out that none of these protocols fulfil the identified requirements. IPsec is not usable because of too exhaustive resource consumption and the limited support of multicast with more than one sender. TLS and SSH would offer good performance, but completely lack multicast support. In principle, the analysed VPN implementation OpenVPN can secure multicast, but only with the use of a centralised server. Besides other disadvantages of this network architecture, this server cannot be implemented on embedded devices.

Therefore, Chapter 4 presents a protocol to secure the communication on the IP backbone of a KNX network. Within this protocol, the communicating entities can authenticate themselves. After the key distribution, which is secured with asymmetric cryptographic algorithms, the communication parties can use the common key set to secure the payload during operation.

A main advantage of the protocol is its flexibility. Due to the used protocol architecture, it is applicable to available HBA standards without the need of an adaption of existing

HBA protocols. [8] showed, that the proposed protocol is also usable in combination with other HBA standards like LonWorks and BACnet. While this generic concept was tailored with HBA networks (especially KNX) in mind, it is also applicable to other application domains (e.g., industrial automation).

Besides this theoretical part, Chapter 5 of this thesis presents a proof-of-concept implementation, which shows the feasibility of the protocol. The prototype was equipped with an ARM7 embedded CPU, running at 18 MHz. The external libraries MIRACL, uIP and the TP1 stack are used for common tasks. Within this thesis, the uIP stack has been modified in order to support IP multicast. This patch has been contributed on the uIP mailing list. The running proof-of-concept implementation offers a good performance, suitable for real world applications.

In the actual version of the protocol, every participant has to fetch the key set from a single coordinator. In bigger networks, especially rekeying scenarios could become problematic, since the coordinator has to handle several requests at the same time. A possible extension could use better key distribution strategies for large networks, like a tree based distribution algorithm. Another possible enhancement concerning rekeying could be the use of an intermediate common key. Instead of the actual key set, the coordinator sends out an intermediate common key. Like in TLS[1], this key will be used by the participants to calculate the actual key set. Every time a rekeying is necessary, the coordinator only has to initiate a recalculation of a new key set.

The proof-of-concept implementation presented in Chapter 5 uses a fixed set of domain parameters, recommend by the NIST (cf. Section 2.4.2). While the used parameters are known to be secure up to know, it may be possible that vulnerabilities against this class of parameters will be found in future. To guard against this problem, elliptic curves could be selected and verified at run time (cf. [23]).

Another possible enhancement concerns the seed of the PRNG. As mentioned in Section 5.1.2, the source for the seed may not be good enough for productive use. Instead other external events like user interactions or network events could be used to increase the quality of the seed.

[1] In TLS, both parties agree on a common pre-master secret and calculate the master secret out of the pre-master secret.

Appendix A

Acronyms

ADC Analog/Digital Converter
AES Advanced Encryption Standard
AH Authentication Header
ARP Adress Resolution Protocol
ASCII American Standard Code for Information Interchange
CA Certification Authority
CBC Cipher Block Chaining
CCM Counter with CBC-MAC
CFB Cipher Feedback
CMAC Cipher-based MAC
CTR Counter
DES Data Encryption Standard
DH Diffie-Hellman (key exchange algorithm)
DLP Discrete Logarithm Problem
DoS Denial-of-Service
DSA Digital Signature Algorithm
ECB Electronic Codeblock
ECC Elliptic Curve Cryptography
ECDH Elliptic Curve Diffie-Hellman
ECDLP Elliptic Curve Discrete Logarithm Problem
ECDSA Elliptic Curve Digital Signature Algorithm
ECIES Elliptic Curve Integrated Encryption Scheme
EIA Electronic Industries Alliance
EIB Europäischer Installationsbus

ESP Encapsulated Security Payload
ETS Engineering Tool Software
GCM Galois/Counter Mode
GDH Group Diffie-Hellman
HMAC Keyed-Hash Message Authentication Code
HBA Home and Building Automation
HTTP Hypertext Transfer Protocol
HVAC heating, ventilation and air conditioning
IETF Internet Engineering Task Force
IFP Integer Factorisation Problem
IKE Internet Key Exchange
IP Internet Protocol
IPv4 Internet Protocol Version 4
IPv6 Internet Protocol Version 6
IPsec Internet Protocol Security
ISO International Organisation for Standardisation
IT Information Technology
IV Initialisation Vector
KDF Key Derivation Function
LAN Local Area Network
LFSR Linear Feedback Shift Register
LSB Least Significant Bit
MAC Message Authentication Code
MII Media Independent Interface
MIRACL Multiprecision Integer and Rational Arithmetic C Library
NAT Network Address Translation
NIST National Institute of Standards and Technology
OFB Output Feedback
OSI Open Systems Interconnection reference model
PDF Portable Document Format
PIN Personal Identification Number
PL Power Line
PRNG Pseudo Random Number Generator
RF Radio frequency
RFC Request for Comments

RSA Cryptographic Algorithm by Ron Rivest, Adi Shamir and Leonard Adleman
RTOS Real Time Operating System
SA Security Association
SECG Standards for Efficient Cryptography Group
SHA Secure Hash Algorithm
SHS Secure Hash Standard
SSH Secure Shell
SSL Secure Sockets Layer
TAN Transaction authentication number
TCP Transmission Control Protocol
TLS Secure Sockets Layer
TP Twisted Pair
TTL Time To Live
TTP Trusted Third Party
UDP User Datagram Protocol
USB Universal Serial Bus
VPN Virtual Private Network
XOR Exclusive OR $\oplus$ (logical operation)
WEP Wired Equivalent Privacy
WPA Wi-Fi Protected Access

Appendix B

Bibliography

[1] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, ''Security in Networked Building Automation Systems,'' in *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, Jun. 2006, pp. 283–292.

[2] ''KNX specification,'' Version 1.1, 2004.

[3] ''KNXnet/IP system specification,'' Version 1.3, 2008.

[4] ''KNX Journal,'' Jan. 2008, pages 2, 11–16.

[5] T. Flanitzer, ''Security mechanisms for low-end embedded systems,'' Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, 2008.

[6] W. Granzer, G. Neugschwandtner, and W. Kastner, ''EIBsec: A Security Extension to KNX/EIB,'' in *Proc. Konnex Scientific Conference*, 11 2006.

[7] W. Granzer, ''Security in Networked Building Automation Systems,'' Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group, Nov. 2005. [Online]. Available: https://www.auto.tuwien.ac.at/thesis/pdf/THESIS0001.pdf

[8] W. Granzer, D. Lechner, F. Praus, and W. Kastner, ''Securing IP Backbones in Building Automation Networks,'' in *Proc. 7th IEEE International Conference on Industrial Informatics (INDIN '09)*, Jul. 2009.

[9] ''Control network protocol specification,'' ANSI/EIA/CEA 709.1, 1999.

[10] ''BACnet – a data communication protocol for building automation and control networks,'' ANSI/ASHRAE 135, 2008.

[11] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*, 2nd ed. New York: Wiley, 1996.

[12] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, ''SPINS: security protocols for sensor networks,'' in *Proc. 7th Annual Intl. Conf. on Mobile Computing and Networking (MobiCom)*, 2001, pp. 189–199.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 5th ed. CRC Press, 2001.

[14] National Institute of Standards and Technology, ''Block Cipher Modes,'' 03 2009. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html

[15] ''Advanced encryption standard (AES),'' FIPS PUB 197, National Institute of Standards and Technology, 2001.

[16] B. Schneier, *Schneier on Security*. Wiley, 9 2008.

[17] S. R. Fluhrer, I. Mantin, and A. Shamir, ''Weaknesses in the key scheduling algorithm of rc4,'' in *SAC '01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*. London, UK: Springer-Verlag, 2001, pp. 1–24.

[18] W. Diffie and M. E. Hellman, ''New directions in cryptography,'' *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976. [Online]. Available: citeseer.ist.psu.edu/diffie76new.html

[19] R. L. Rivest, A. Shamir, and L. Adleman, ''A method for obtaining digital signatures and public-key cryptosystems,'' *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[20] D. Boneh, T. R. Cryptosystem, I. R. Rivest, A. Shamir, L. Adleman, and W. Rst, ''Twenty years of attacks on the rsa cryptosystem,'' *Notices of the AMS*, vol. 46, pp. 203–213, 1999.

[21] D. Boneh, A. Joux, and P. Q. Nguyen, ''Why textbook elgamal and rsa encryption are insecure,'' in *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2000, pp. 30–43.

[22] T. El Gamal, ''A public key cryptosystem and a signature scheme based on discrete logarithms,'' in *Proceedings of CRYPTO 84 on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

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

[24] S. Bartolini, I. Branovic, R. Giorgi, and E. Martinelli, ''A performance evaluation of arm isa extension for elliptic curve cryptography over binary finite fields,'' in *IEEE 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD-04)*, Foz do Iguacu, Brasil, Oct. 2004, pp. 238–245.

[25] M. Aydos, T. Yantk, and Ç. K. Koç, ''A high-speed ecc-based wireless authentication on an arm microprocessor,'' in *ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2000, p. 401.

[26] Standards for Efficient Cryptography Group, ''SEC 2: Recommended Elliptic Curve Domain Parameters,'' SEC 2, 09 2000, version 1.0. [Online]. Available: http://www.secg.org/download/aid-385/sec2_final.pdf

[27] National Institute of Standards and Technology, ''Recommended elliptic curves for federal government use,'' 07 1999. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

[28] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, ''Collisions of sha-0 and reduced sha-1,'' in *EUROCRYPT*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 36–57. [Online]. Available: http://dx.doi.org/10.1007/11426639_3

[29] X. Wang, Y. L. Yin, and H. Yu, ''Finding collisions in the full sha-1,'' in *CRYPTO*, ser. LNCS, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 17–36. [Online]. Available: http://dx.doi.org/10.1007/11535218_2

[30] ''Secure hash standard,'' FIPS PUB 180-2, National Institute of Standards and Technology, Aug. 2002.

[31] H. Krawczyk, M. Bellare, and R. Canetti, ''HMAC: Keyed-Hashing for Message Authentication,'' RFC 2104 (Informational), Feb. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt

[32] ''Digital signature standard,'' FIPS PUB 186, National Institute of Standards and Technology, May 1994.

[33] ''Digital signature standard,'' FIPS PUB 186-3, National Institute of Standards and Technology, 06 2009.

[34] D. Johnson, A. Menezes, and S. Vanstone, ''The Elliptic Curve Digital Signature Algorithm (ECDSA),'' *International Journal of Information Security*, vol. 1, no. 1, 08 2001.

[35] Standards for Efficient Cryptography Group, ''SEC 1: Elliptic Curve Cryptography,'' SEC 1, 09 2000, version 1.0. [Online]. Available: http://www.secg.org/download/aid-385/sec1_final.pdf

[36] L. Dorrendorf, Z. Gutterman, and B. Pinkas, ''Cryptanalysis of the windows random number generator,'' in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 476–485.

[37] L. Bello. (2008, 05) Dsa-1571-1 openssl – predictable random number generator. [Online]. Available: http://www.debian.org/security/2008/dsa-1571.en.html

[38] National Institute of Standards and Technology, ''Random Number Generation,'' 03 2009. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/random_number.html

[39] D. E. and 3rd, J. Schiller, and S. Crocker, ''Randomness Requirements for Security,'' RFC 4086 (Best Current Practice), Jun. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4086.txt

[40] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, ''The Kerberos Network Authentication Service (V5),'' RFC 4120 (Proposed Standard), Jul. 2005, updated by RFCs 4537, 5021. [Online]. Available: http://www.ietf.org/rfc/rfc4120.txt

[41] M. Steiner, G. Tsudik, and M. Waidner, ''Diffie-hellman key distribution extended to group communication,'' in *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*. New York, NY, USA: ACM, 1996, pp. 31–37.

[42] I. Chatzigiannakis, E. Konstantinou, V. Liagkou, and P. Spirakis, ''Design, analysis and performance evaluation of group key establishment in wireless sensor networks,'' *Electron. Notes Theor. Comput. Sci.*, vol. 171, no. 1, pp. 17–31, 2007.

[43] Internet Engineering Task Force, ''IETF Home Page,'' 01 2009. [Online]. Available: http://www.ietf.org

[44] K. G. Paterson and A. K. Yau, ''Cryptography in theory and practice: The case of encryption in ipsec,'' Cryptology ePrint Archive, Report 2005/416, 2005, http://eprint.iacr.org/.

[45] C. Kaufman, ''Internet Key Exchange (IKEv2) Protocol,'' RFC 4306 (Proposed Standard), Dec. 2005, updated by RFC 5282. [Online]. Available: http://www.ietf.org/rfc/rfc4306.txt

[46] T. Hardjono and B. Weis, ''The Multicast Group Security Architecture,'' RFC 3740 (Informational), Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3740.txt

[47] N. Ferguson and B. Schneier, ''A cryptographic evaluation of ipsec,'' Counterpane Internet Security, Inc, 12 2003.

[48] T. Dierks and E. Rescorla, ''The Transport Layer Security (TLS) Protocol Version 1.2,'' RFC 5246 (Proposed Standard), Aug. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[49] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz, ''Sizzle: A standards-based end-to-end security architecture for the embedded internet,'' in *Proc. of the 3rd IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 247–256.

[50] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, ''Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS),'' RFC 4492 (Informational), May 2006, updated by RFC 5246. [Online]. Available: http://www.ietf.org/rfc/rfc4492.txt

[51] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis, ''A Framework for IP Based Virtual Private Networks,'' RFC 2764 (Informational), Feb. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2764.txt

[52] (2008, Nov.) OpenVPN. [Online]. Available: http://openvpn.net

[53] T. Ylonen and C. Lonvick, ''The Secure Shell (SSH) Protocol Architecture,'' RFC 4251 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4251.txt

[54] ——, ''The Secure Shell (SSH) Transport Layer Protocol,'' RFC 4253 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4253.txt

[55] ——, ''The Secure Shell (SSH) Authentication Protocol,'' RFC 4252 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4252.txt

[56] ——, ''The Secure Shell (SSH) Connection Protocol,'' RFC 4254 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4254.txt

[57] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[58] T. Aura, P. Nikander, and J. Leiwo, ''Dos-resistant authentication with client puzzles,'' in *Revised Papers from the 8th International Workshop on Security Protocols*. London, UK: Springer-Verlag, 2001, pp. 170–177.

[59] Siemens, ''Technical Data EIB-TP-UART-IC,'' 2001, version D.

[60] G. Combs. (2008, Nov.) Wireshark website. [Online]. Available: http://www.wireshark.org

[61] H. Weillechner and D. Lechner. (2009, 03) Wireshark knxnet/ip plugin. [Online]. Available: https://www.auto.tuwien.ac.at/a-lab/wireshark-plugin-for-knxnet/ip.html

[62] A. Dunkels. (2008, Oct.) uip website. [Online]. Available: http://www.sics.se/~adam/uip

[63] (2008, Oct.) Shamus software ltd. [Online]. Available: http://www.shamus.ie

[64] W. Granzer and W. Kastner, ''BACnet over KNX,'' in *Proc. Konnex Scientific Conference*, 11 2007.

[65] B. Moeller. (2008, Nov.) Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures. [Online]. Available: http://www.openssl.org/~bodo/tls-cbc.txt

Appendix C

Code snippets

C.1 HMAC

Listing C.1 shows the implementation of the HMAC function to demonstrate the use of MIRACL hashing functions.

Listing C.1: HMAC code

```
void hmac(char* key, char* text, int textlen, int maclen, char* mac)
{
  int i;
  /// initialise the sha-instances
  sha256 inner_sha, outer_sha;
  shs256_init(&inner_sha);
  shs256_init(&outer_sha);
  /// calculate the inner hash-value
  for (i=0;i<maclen;i++)
  {
    shs256_process(&inner_sha,(key[i]^ipad));
  }
  for (i=0;i<textlen;i++)
  {
    shs256_process(&inner_sha,text[i]);
  }
  shs256_hash(&inner_sha,mac);
  /// calculate the outer hash-value
  for (i=0;i<maclen;i++)
  {
    shs256_process(&outer_sha,(key[i]^opad));
```

```
  }
  for (i=0;i<maclen;i++)
  {
    shs256_process(&outer_sha,mac[i]);
  }
  shs256_hash(&outer_sha,mac);
  return;
}
```

C.2 DH

Listing C.2 is another example for the integration of MIRACL functions. In contrast to Listing C.1, more advanced ECC functions are engaged. Since all variables are static (cf. Section 5.1.2), the space for these variables have to be known at compile time. In Line 4, a pointer to the elliptic curve point struct is defined. Line 5 reserves the needed memory for exactly 1 elliptic curve point. It's real size (i.e. how many "characters" are reserved) depends on the initialisation of the MIRACL system, where number sizes and the internal representation format are set. Line 6 initialises the memory with zeros to be usable by the MIRACL system.

Since all numbers are sent using point compression (cf. Section 5.1.3), the public key has to be decompressed first. Afterwards this point is multiplied with the given private key to derive the shared secret key.

Listing C.2: DH code

```
BOOL _dh2(big priv_key, big pub_key, int pub_key_lsb, big z)
{
  BOOL valid;
  epoint *e_pub_rcv;
  char mem1[MR_ECP_RESERVE(1)];
  memset(mem1,0,MR_ECP_RESERVE(1));
  // extract public key
  e_pub_rcv=epoint_init_mem(_mip,mem1,0);
  valid=epoint2_set(_mip,pub_key,pub_key,pub_key_lsb,e_pub_rcv);  /*
      decompress public key */
  if(!valid)
  {
    printf("DH: INVALID PUBLIC KEY\n\r");
```

```
    return FALSE;
  }
  // multiplicate the point with the private key
  ecurve2_mult(_mip,priv_key,e_pub_rcv,e_pub_rcv);
  if (point_at_infinity(e_pub_rcv))
  {
    printf("point at infitiy!\n\r");
    return FALSE;
  }
  epoint2_get(_mip,e_pub_rcv,z,z);
  return TRUE;
}
```

C.3 Frame building

Listing C.3 is used to demonstrate the symmetric encryption of a frame. The function builds a frame as shown in Figure 5.11. As described, the whole message will be secured by a MAC field and some fields are additionally encrypted.

Line 26 sets the IV for the encryption operation. After the pieces of the frame are copied into a buffer and the HMAC tag is calculated, the encryption takes place. As already described in Section 5.1.4, the block encryption has to be performed several times. Line 50 is the loop which executes the MIRACL encryption operation in Line 52 multiple times to encrypt the desired fields.

Listing C.3: Frame building and symmetric encryption

```
int comm_build_frame(char* buffer, int framelen, uint32_t counter)
{
  /// calculate padding length
  char padlen = COMM_BLOCKLEN-((framelen+COMM_MACLEN+COMM_PADLENLEN) %
      COMM_BLOCKLEN);
  if(padlen == COMM_BLOCKLEN) padlen = 0;
  /// check the size of the resulting frame
  if(COMM_LENLEN+COMM_IVLEN+framelen+COMM_MACLEN+padlen+COMM_PADLENLEN >=
      COMM_FBLEN)
    return EC_BUFTOOSMALL;
  /// move buffer COMM_IVLEN+COMM_LENLEN bytes backwards
  int pos;
  for (pos=framelen-1; pos>=0; pos--)
```

```
{
  buffer[COMM_LENLEN+COMM_IVLEN+pos] = buffer[pos];
}
framelen+=COMM_LENLEN+COMM_IVLEN;
/// append padding and padding length to frame
int padfil;
for (padfil=padlen+COMM_PADLENLEN-1; padfil>=0; padfil--)
{
  // write something into buffer (padding) + the padlength at the end (
      in every case)
  buffer[framelen+COMM_MACLEN+padfil]=padlen;
}
int macpos = framelen;
framelen+=COMM_MACLEN+padlen+COMM_PADLENLEN;
/// copy IV into buffer
aes_getreg(&_a_encr,buffer+COMM_LENLEN);
/// set length
framelen-=(COMM_LENLEN);
buffer[1] = (char) (framelen & 0xFF);
buffer[0] = (char) ((framelen>>8) & 0xFF);
framelen+=(COMM_LENLEN);
/// calculate and set MAC
// MAC = HMAC(counter||length||iv||data||padding||padlen)
char mac[COMM_MACLEN];
comm_framebuffer text2mac;  // buffer which will be used for mac
    calculation
// set counter
text2mac[3] = (char) (counter & 0xFF);
text2mac[2] = (char) ((counter>>8) & 0xFF);
text2mac[1] = (char) ((counter>>16) & 0xFF);
text2mac[0] = (char) ((counter>>24) & 0xFF);
// copy length||iv||data into mac-buffer
memcpy(text2mac+COMM_COUNTERLEN,buffer,macpos);
// copy padding||padlen into mac-buffer
memcpy(text2mac+COMM_COUNTERLEN+macpos,buffer+macpos+COMM_MACLEN,padlen+
    COMM_PADLENLEN);
hmac(_mac_key, text2mac, COMM_COUNTERLEN+framelen-COMM_MACLEN,
    COMM_MACLEN, mac);
memcpy(buffer+macpos,mac,COMM_MACLEN); //insert the calculated MAC
/// encrypt now
int encr_len = framelen-COMM_IVLEN-COMM_LENLEN;
int encr_num;
```

```
  for(encr_num=0; encr_num < encr_len/COMM_BLOCKLEN; encr_num++)
  {
    aes_encrypt(&_a_encr,buffer+COMM_LENLEN+COMM_IVLEN+(COMM_BLOCKLEN*
        encr_num));
  }
  return framelen;
}
```

C.4 uIP patches

Listing C.4: uIP multicast patch

```
--- uip_orig.c  2009-03-25 11:27:56.000000000 +0100
+++ uip.c 2009-03-25 11:24:52.000000000 +0100
@@ -919,7 +919,12 @@
     /* Check if the packet is destined for our IP address. */
 #if !UIP_CONF_IPV6
+//If multicast is enabled, check if this is a multicast packet
+#if UIP_CONF_MULTICAST
+     if(!uip_ipaddr_cmp(BUF->destipaddr, uip_hostaddr) && (BUF->
    destipaddr[0] < HTONS(0xe000) || BUF->destipaddr[0] > HTONS(0xefff)))
    {
+#else
      if(!uip_ipaddr_cmp(BUF->destipaddr, uip_hostaddr)) {
+#endif /*UIP_CONF_MULTICAST*/
       UIP_STAT(++uip_stat.ip.drop);
       goto drop;
     }
@@ -1116,7 +1121,14 @@
         UDPBUF->srcport == uip_udp_conn->rport) &&
        (uip_ipaddr_cmp(uip_udp_conn->ripaddr, all_zeroes_addr) ||
  uip_ipaddr_cmp(uip_udp_conn->ripaddr, all_ones_addr) ||
+//If multicast is compiled, and destip = rip and that ip is a multicast
    ip
+#if UIP_CONF_MULTICAST
+    (uip_ipaddr_cmp(BUF->destipaddr, uip_udp_conn->ripaddr) && !(BUF->
    destipaddr[0] < HTONS(0xe000) || BUF->destipaddr[0] > HTONS(0xefff)))
    ||
+#endif /*UIP_CONF_MULTICAST*/
  uip_ipaddr_cmp(BUF->srcipaddr, uip_udp_conn->ripaddr))) {
```

```
+#if UIP_CONF_MULTICAST
+     uip_ipaddr_copy(uip_udp_conn->srcaddr,BUF->srcipaddr);
+#endif /*UIP_CONF_MULTICAST*/
      goto udp_found;
     }
   }
```

Listing C.5: uIP multicast patch (header)

```
--- uip_orig.h  2009-03-25 11:14:32.000000000 +0100
+++ uip.h 2009-03-25 11:19:38.000000000 +0100
@@ -1213,6 +1213,10 @@
   u16_t rport;        /**< The remote port number in network byte order.
        */
   u8_t  ttl;          /**< Default time-to-live. */
+#if UIP_CONF_MULTICAST
+  uip_ipaddr_t srcaddr;   /**< For multicast packages, this is the
    sourceaddress (ripaddr contains the multicast address). */
+#endif /*UIP_CONF_MULTICAST*/
+
   /** The application state. */
   uip_udp_appstate_t appstate;
 };
```

Listing C.6: uIP multicast patch (ARP)

```
--- uip_arp_orig.c  2008-01-23 12:28:50.000000000 +0100
+++ uip_arp.c 2008-11-17 17:54:02.000000000 +0100
@@ -102,6 +102,9 @@
   u8_t time;
 };
+static const struct uip_eth_addr multicast_upper_ethaddr =
+  {{0x01,0x00,0x5e,0x00,0x00,0x00}};
+
 static const struct uip_eth_addr broadcast_ethaddr =
   {{0xff,0xff,0xff,0xff,0xff,0xff}};
 static const u16_t broadcast_ipaddr[2] = {0xffff,0xffff};
@@ -365,6 +368,12 @@
   /* First check if destination is a local broadcast. */
   if(uip_ipaddr_cmp(IPBUF->destipaddr, broadcast_ipaddr)) {
     memcpy(IPBUF->ethhdr.dest.addr, broadcast_ethaddr.addr, 6);
+  } else if(IPBUF->destipaddr[0] >= HTONS(0xe000) || IPBUF->destipaddr[0]
    <= HTONS(0xefff)) {
+    /* is a multicast */
```

```
+    // build ethernet MAC address
+    memcpy(IPBUF->ethhdr.dest.addr, multicast_upper_ethaddr.addr, 3);
+    memcpy((IPBUF->ethhdr.dest.addr)+4, (IPBUF->destipaddr)+1, 2);
+    IPBUF->ethhdr.dest.addr[3]=(IPBUF->destipaddr[0]) & HTONS(0x007F);
   } else {
     /* Check if the destination address is on the local network. */
     if(!uip_ipaddr_maskcmp(IPBUF->destipaddr, uip_hostaddr, uip_netmask))
          {
@@ -376,7 +385,10 @@
       /* Else, we use the destination IP address. */
       uip_ipaddr_copy(ipaddr, IPBUF->destipaddr);
     }
+    /* in case of multicast -> bypass arp-lookup */
+    if(IPBUF->destipaddr[0] < HTONS(0xe000) || IPBUF->destipaddr[0] >
     HTONS(0xefff)) {
+      /* is a multicast -> bypass arp-lookup */
     for(i = 0; i < UIP_ARPTAB_SIZE; ++i) {
       tabptr = &arp_table[i];
       if(uip_ipaddr_cmp(ipaddr, tabptr->ipaddr)) {
@@ -411,8 +423,9 @@
     /* Build an ethernet header. */
     memcpy(IPBUF->ethhdr.dest.addr, tabptr->ethaddr.addr, 6);
   }
+  }
   memcpy(IPBUF->ethhdr.src.addr, uip_ethaddr.addr, 6);
   IPBUF->ethhdr.type = HTONS(UIP_ETHTYPE_IP);
   uip_len += sizeof(struct uip_eth_hdr);
```

C.5 Other examples

Listing C.7 demonstrates the write operation into the nonvolatile flash memory. During the operation, the interrupt for the periodical timer must be deactivated. This example writes some dummy data into the flash area. After the write progress was successful, the written data can be access by using the pointer `inFlash`.

Listing C.7: Flash memory write example

```
// disable TC, ...
clock_disable();
// initialise flash
FLASHD_Initialize(BOARD_MCK);
```

```
// unlock page
if(FLASHD_Unlock(AT91C_IFLASH, AT91C_IFLASH + AT91C_IFLASH_SIZE, 0, 0)!=0)
     {
  printf("Error while trying to unlock page\r\n");
}
// get pointer to flash location
void* inFlash;
inFlash = (void*) AT91C_IFLASH + AT91C_IFLASH_SIZE -
    AT91C_IFLASH_PAGE_SIZE;
// reserve temporary buffer and fill with numbers
unsigned int pageBuffer[AT91C_IFLASH_PAGE_SIZE / (sizeof(unsigned int))];
int i;
for (i=0; i < (AT91C_IFLASH_PAGE_SIZE / sizeof(unsigned int)); i++) {
    pageBuffer[i] = 1 << (i % 32);
}
// write into flash
if(FLASHD_Write((unsigned int)inFlash, pageBuffer, AT91C_IFLASH_PAGE_SIZE)
     != 0) {
  printf("Error when trying to write page\n\r");
}
// enable interrupts
clock_enable();
```