FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

KNX for Safety Critical Environments

Building a secure and dependable Layer

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Harald Glanzer
Matrikelnummer 0727156

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: ao.Univ.-Prof. Dipl.-Ing. Wolfgang Kastner
Mitwirkung: Dr. Lukas Krammer

Wien, 1.12.2014

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

KNX for Safety Critical Environments

Building a secure and dependable Layer

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Harald Glanzer
Registration Number 0727156

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: ao.Univ.-Prof. Dipl.-Ing. Wolfgang Kastner
Assistance: Dr. Lukas Krammer

Vienna, 1.12.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Harald Glanzer
Hardtgasse 25 / 12A, 1190 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

Optional acknowledgements may be inserted here.

Abstract

According to the guidelines of the faculty, an abstract in English has to be inserted here.

Kurzfassung

Hier fügen Sie die Kurzfassung auf Deutsch gemäß den Vorgaben der Fakultät ein.

Contents

CHAPTER 1

Introduction

1.1 General Information

This document is intended as a template and guideline and should support the author in the course of doing the master's thesis. Assessment criteria comprise the quality of the theoretical and/or practical work as well as structure, content and wording of the written master's thesis. Careful attention should be given to the basics of scientific work (e.g., correct citation).

1.2 Organizational Issues

A master's thesis at the Faculty of Informatics has to be finished within six months. During this period regular meetings between the advisor(s) and the author have to take place. In addition, the following milestones have to be fulfilled:

1. Within one month after having fixed the topic of the thesis the master's thesis proposal has to be prepared and must be accepted by the advisor(s). The master's thesis proposal must follow the respective template of the dean of academic affairs. Thereafter the proposal has to be applied for at the deanery. The necessary forms may be found on the web site of the Faculty of Informatics. `http://www.informatik.tuwien.ac.at/dekanat/formulare.html`

2. Accompanied with the master's thesis proposal, the structure of the thesis in terms of a table of contents has to be provided.

3. Then, the first talk has to be given at the so-called "Seminar for Master Students". The slides have to be discussed with the advisor(s) one week in advance. Attendance of the "Seminar for Master Students" is compulsory and offers the opportunity to discuss arising problems among other master students.

4. At the latest five months after the beginning, a provisional final version of the thesis has to be handed over to the advisor(s).

5. As soon as the provisional final version exists, a first poster draft has to be made. The making of a poster is a compulsory part of the "Seminar for Master Students" for all master studies at the Faculty of Informatics. Drafts and design guidelines can be found at `http://www.informatik.tuwien.ac.at/studium/richtlinien`.

6. After having consulted the advisor(s) the second talk has to be held at the "Seminar for Master Students".

7. At the latest six months after the beginning, the corrected version of the master's thesis and the poster have to be handed over to the advisor(s).

8. After completion the master's thesis has to be presented at the "epilog". For detailed information on the epilog see:
`http://www.informatik.tuwien.ac.at/studium/epilog`

1.3 Structure of the Master's Thesis

If the curriculum regulates the language of the master's thesis to be English (like for "Business Informatics"), the thesis has to be written in English. Otherwise, the master's thesis may be written in English or in German. The structure of the thesis is predetermined. The table of contents is followed by the introduction and the main part, which can vary according to the content. The master's thesis ends with the bibliography (compulsory) and the appendix (optional).

- Cover page
- Acknowledgements
- Abstract of the thesis in English and German
- Table of contents
- Introduction
  - motivation
  - problem statement (which problem should be solved?)
  - aim of the work
  - methodological approach
  - structure of the work
- State of the art / analysis of existing approaches
  - literature studies

  - analysis
  - comparison and summary of existing approaches
- Methodology
  - used concepts
  - methods and/or models
  - languages
  - design methods
  - data models
  - analysis methods
  - formalisms
- Suggested solution/implementation
- Critical reflection
  - comparison with related work
  - discussion of open issues
- Summary and future work
- Appendix: source code, data models, ...
- Bibliography

CHAPTER 2

KNX

2.1 Introduction to KNX

KNX implements a specialized form of automated process control, dedicated to the needs of home and building automation. KNX emerged from 3 leading standards:

- EIB
- EHS
- BATIBUS

Datapoints

Datapoints represent the process- and control variables of the system, and can be of inputttype, outputttype, diagnostic data, parameters or others. To achieve interoperability, Datapoints are grouped into functional blocks, implementing standardized datapoint types.

2.2 KNX Layers

The *Open Systems Interconnection Model* OSI standardizes the communication between different, independent systems by grouping the needed functions into 7 sublayers to provide interchangeability and abstraction. KNX implements this model, omitting layers 5 and 6, as shown in figure 2.2. Data from applications are directly passed to the transport layer in a transparent way, and vice versa.

stack.png stack.png

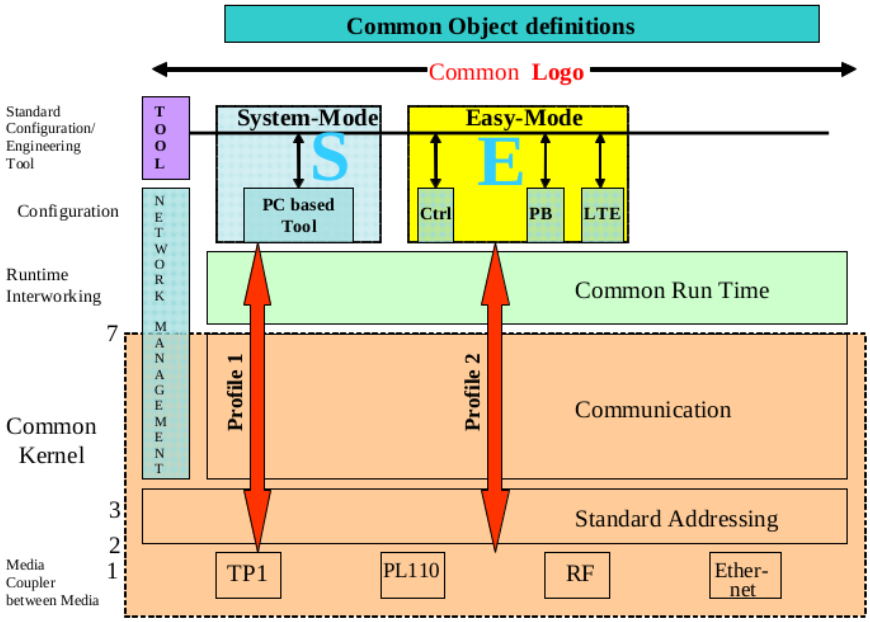


Figure 2.1: The KNX Stack

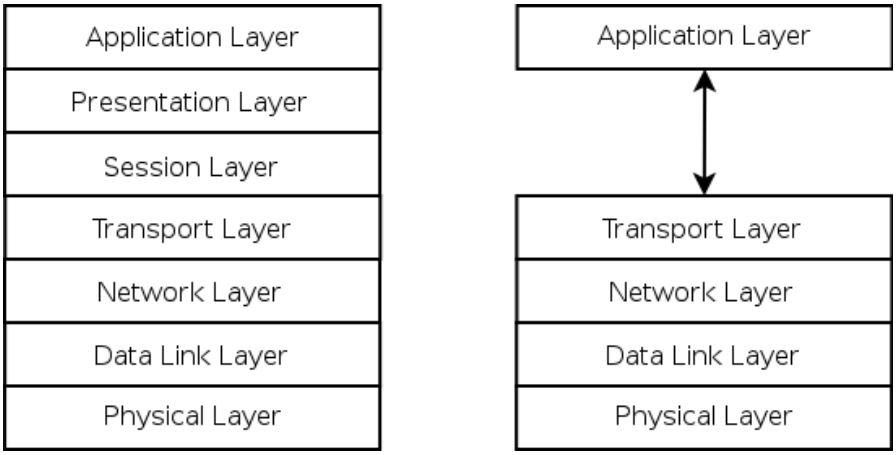


Figure 2.2: OSI Layer Model, compared to the KNX Model

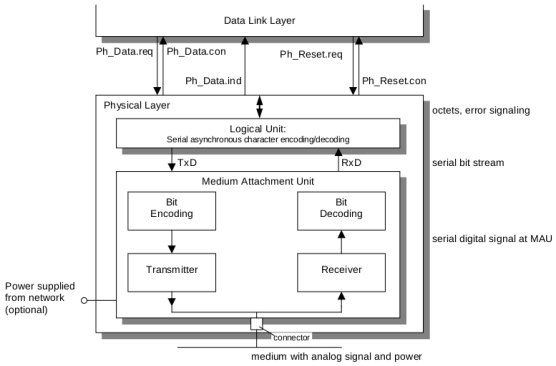


Figure 2.3: Logical Structure of TP1

Physical Layer

- TP1 was inherited from EIB and is the basis medium, consisting of a twisted pair cabeling. Data and power can be transmitted with one pair, so low-power devices can be fed over the bus. Data transfer is done asynchronously, with bidirectional, half-duplex communication and a datarate of 9600 bit/sec. TP1 uses collision avoidance, and allows all topologies beside rings.

  Because this work is base on the TP1 - part of KNX only, this physical layer is explained in more detail in the next chapter.

- PL110, which was also inherited from EIB, uses the power line for communications. The carrier uses spread frequency shift keying, also for bidirectional, half duplex communication with a even lower data rate of 1200 bit/sec

- RF is used for short range wireless communication at 868,3 MHz.

- IP Gateway FIXME

TP1

The accurate name for this medium is 'Physical Layer type Twisted Pair', with variants PhL TP1-64 and PhL TP1-256, which is backward compatible to the former one.

The logical structure of the TP1 layer is shown in figure2.3
A logical '1' is regarded as the idle state of the bus, so the transmitter of the MAU(Medium Access Unit) is disabled when sending a '1', so the analog signal on the bus consists only of the DC part.
A logical '0' is then defined as the voltage drop for a specified duration of t_{active}, after which the voltage can increase over DC level, see figure2.4

TP1 implements CSMA/CA, so devices will listen to the bus and should only begin sending when the bus is idle. In the case of a simultaneous transmission start, a logical '1' of one device

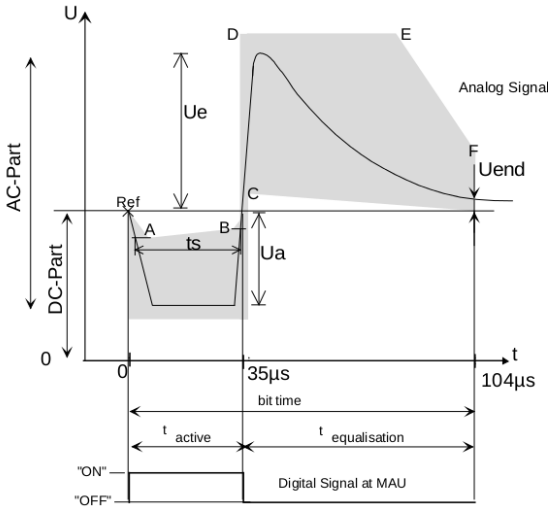


Figure 2.4: Logical 0

Control field

7	6	5	4	3	2	1	0	
Frame type		repeat flag		priority	priority			
FT	0	r	1	p1	p0	0	0	L_Data-Frame
1	1	1	1	0	0	0	0	L_Poll_Data-Frame
x	x	0	0	x	x	0	0	Acknowledgement Frame

FT = 0 : L_Data_Extended Frame
FT = 1 : L_Data_Standard Frame

r = 0 : repeated L_Data-Frame
r = 1 : not repeated L_Data-Frame

p1	p0	
0	0	system priority
1	0	urgent priority
0	1	normal priority
1	1	low priority

Figure 2.5: Control Field

will eventually be overwritten by a logical ’0’ of the other device. The overruled sender will detect this by continuously checking the state of the bus and has to stop transmission.

Data Link Layer

There are defines 3 frame formats which are allowed, each with the MSB sent first. Every frame starts with a control field which determines the exact type of the following frame, as shown in figure2.5. In case of the simultaneous start of a transmission, the kind of frame sent, the priority bits of the control field set or at the latest the source address cause a transmit abortion by the CSMA/CA mechanism used, as explained above.

FIXME reihenfolge der frames priority bits are bits 4, 5 –> leading bits <–> priority??

1. extended frames
2. dataframes

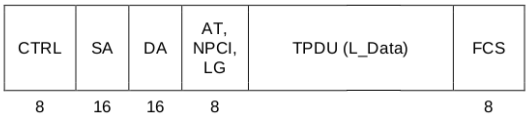


Figure 2.6: Standard Frame

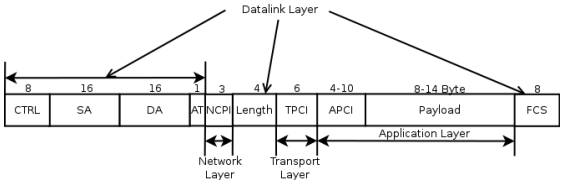


Figure 2.7: Standard Frame, in detail

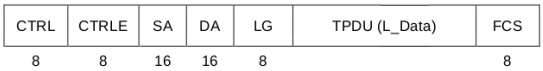


Figure 2.8: Extended Frame

3. repeat frames

- L_Data Frame(Standard or Extended)
- L_Poll_Data Frame
- Acknowledge Frame

L_Data_Frame

This frame can have two formats: standard2.6 and extended2.8. Both start with the control field, for extended frames an extended control field follows. After that, for booth frames sender address(SA) and destination address(DA), each 2 byte, follow. The next byte has different meanings: for standard frames, 1 address type bit, 3 and 4 bits of length information, resulting in an maximum payload of 15 bytes. The extended frame reserves 8 bit of length information, allowing a maximum payload of 248 FIXME bytes(0xFF is defined as escape code). FIXME

TPCI(transport layer protocol control information) bit - point to point connection. APCI(application layer protocol control information) bit - read / write / response

For booth formats, a check byte completes the frame, calculating a bitwise NOT XOR function over all bytes.

L_Poll_Data Frame

These frames serve as data requests for a maximum of 15 bytes(payload of a standard data frame) and start with a control field, as defined, followed by the 2 byte source address of the

sender(called Poll_Data Master). The following 2 byte destination address is used to address up to 15 poll slaves, all belonging to the same poll group. The number of exptected bytes and the check octet follow.

Poll requests are answered by poll slaves just by transmitting the databytes without any special format, sent in the corresponding poll slave slot.

Acknowledge Frame

This frames are used to acknowledge request frames by just sending one byte representing ACK, BUSY or NOT ACK.

Network Layer

Transport Layer

This layer provides services for reliable point-to-point connections, as well as connection-less point-to-point, point-to-domain(multicast) and point-to-all-points(broadcast):

- point-to-point, connectionless
  T_Data_Individual: connection-less, unreliable point-to-point
- point-to-point, connection orientated:
  T_Connect: establish reliable connection to individual address
  T_Data_Connected: send data over established connection
  T_Disconnect: terminate connection to individual address
  This service uses a timer to detect timeouts, and allows up to 3 retransmissions. Prior to sending data, a T_Connect request has to be sent. If the remote device cannot handle a new connection, a T_Disconnect request is returned to the sender. Otherwise, no confirmation or acknowledgment is sent to the remote device.
- point-to-multipoint, connectionless
  T_Data_Group: unreliable multicast to group address
- point-to-all-points, connectionless
  T_Data_Broadcast: unreliable broadcast to all devices of a domain
- point-to-all-points, connectionless
  T_Data_SystemBroadcast: unreliable broadcast to all devices of a domain FIXME: unterschied Broadcast-SystemBroadcast

Session Layer

This layer is left empty in KNX and just forwards data transparently from the lower level to the upper level, and vice versa.

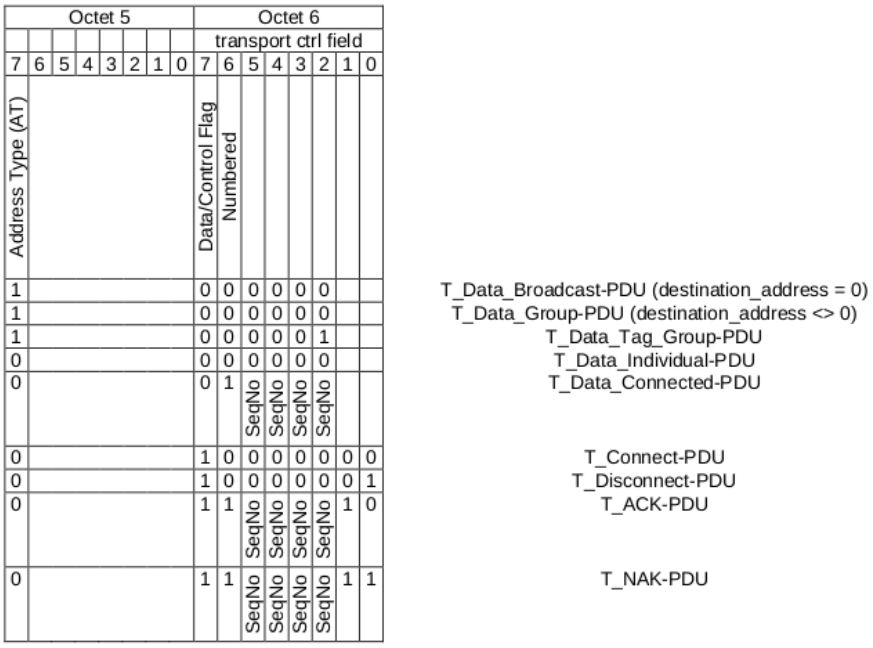


Figure 2.9: Flags used at the Transport Layer

Presentation Layer

This layer is left empty in KNX and just forwards data transparently from the lower level to the upper level, and vice versa.

Application Layer

Topologies

TP1-64 allows up to 64, and TP1-256 up to 256 devices connected to one physical segment in a linear, star, tree or mixed topology. The maximum distance between 2 devices in one physical segment is 700m, with all devices in this segment sharing the logical address space of a subnetwork.

Up to 4 physical segments are combined by bridges to form lines, guaranteeing a galvanic separation of these segments and extending the maximum allowed distance between devices, with a maximum of 255 devices per line. While a bridge itself has no individual address, they acknowledge frames received and transmit them on the other side.

Theses lines itself can be combined by routers, forming so called areas, also providing a galvanic separation.

CHAPTER 3

Mathematical Background

- Divisiblity
  $a|b \iff \exists z \in \mathbb{Z} : b = z \times a$
- Integer Division $r \equiv a \bmod b$
  $\forall a, b \in \mathbb{Z}, b > 0 : a = z \times b + r$ with $0 \leq r < b$
- Common Divisor c for $a, b \iff$
  $c|a \wedge c|b$
- Greatest Common Divisor(GCD) d:
  $d|a \wedge d|b$ and whenever $c|a \wedge c|b \rightarrow c|d$
- unlimited number of primes
- every number has exact 1 factorization

3.1 Finite fields

3.2 One Way functions

The idea for this concept was formulated for the first time in the year 1874 by William Stanley Jevons in his book 'The Principles of Science'(page 144).

CHAPTER 4

Cryptopgraphy

Cryptography(classical greek for *kryptôs*, which means *concealed*) is the science of encrypting information. Related to computer engineering, it is the art of hiding information by turning cleartext data into a pseudo-random looking stream or block of bits, called ciphertext, using some kind of *key*. This process is called *encryption*. Unauthorized parties - lacking the used key - should, by looking at the ciphertext, learn absolutely nothing about the hidden cleartext. Authorized parties, on the other hand, are able to retrieve the original data out of the ciphertext by using the key, and thus reversing the encryption. This reversing process is called *decryption*.

This way, cryptography is used to provide these objectives:

- Privacy / Confidentiality: protected data can only be read by authorized parties
- Integrity: ensuring that information has not been altered, or altering can be detected
- Authenticity: the origin of the data is genuine
- Non-Repudiation: prevent an entity from denying commitments or actions how had been granted earlier

FIXME: comutationally secure vs. unconditionally secure, i.e. one time pad(perfect secrecy?) FIXME: MERKLE puzzles

A fundamental goal of cryptography is to provide all of these 4 objectives. Providing these objectives only partially will not result in a secure system. For example, providing privacy and integrity, but no authenticity, may lead to so called 'replay' attacks, as will be shown. FIXME: replay attack

To be able to make rigorous statements, it is important to introduce some basic background knowledge, which is given in the next chapter:

Propabilistic Theory

4.1 History of Cryptography

FIXME:BIBLOGRAPHY: the codebreakers

The evolution of cryptography was no linear process. Ciphers were used independently in different places, were forgotten and disappeared when the corresponding civilization died. Nevertheless, basics are found thousands of years ago, therefore a short time line showing some noteable events is presented below:

- Egypt, about 2000 B.C.: a simple substitution is used to partially replace ordinary hieroglyphs with special ones.
- Sparta, about 500 B.C.: a device called 'skytale' is used to encrypt military messages
- Rome, about 100 B.C.: Caesar uses a simple substitution cipher for military correspondence, replacing every letter of the alphabet with the letter 4 places further down the alphabet
- China, 11th century: for a list of 40 items, the first 40 ideograms of a poem are assigned
- Arabia, 14th century: 7 different ciphers and the frequency analysis of letters are introduced by Ahmas al-Qalqashandi
- Vienna, 17th century: the 'Geheime Kabinets-Kanzlei' routinely intercepts, copies and reseales diplomatic correspondence to embassies, and manages to decrypt a great percentage of the ciphertexts
- England, 19th century: Samuel F.B. Morse invents the Morse Code
- England, 1940: the so called 'Turing Bomb', an electromechanical device, is used to decrypt German Enigma-based ciphertexts
- USA, 1976: Whitfield Diffie and Martin Hellman specify the a protocol for key exchange, based on a public key system developed by Ralph Merkle
- USA, 1977: RSA public key encryption is defined

4.2 Definitions and Basic Assumptions

- $\mathcal{A}$ is a finite set, denoting the alphabet used, for example $\mathcal{A} = \{0, 1\}$
- $\{0, 1\}^n$ denotes the set of all possible strings with length n
- $\mathcal{M}$ is the message space, consisting of all strings that can be built with the underlying alphabet
- $\mathcal{C}$ is the ciphertext space, also consisting of the strings from the alphbet

- $\mathcal{K}$ is called keyspace, also built from the alphabet. Every element $e \in \mathcal{K}$ is called a key and determines the function $\mathcal{M} \to \mathcal{C}$. This function, E_e is called the *encryption function*.

$$ciphertext = E_e(e, cleartext)$$

- For every key $d \in \mathcal{K}$, D_d denotes the function from $\mathcal{C} \to \mathcal{M}$, and is called *decryption function*.

$$cleartext = D_d(d, ciphertext)$$

- The keys e and d are also referred to as *keypair*, written (e, d).
- If it is computationally easy to derive the private key e from the public key d(in most cases $e = d$), the encryption scheme is called *symmetric*, otherwise the scheme is called *asymmetric*.
- A cipher(also called *encryption scheme*) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ is a pair of *efficient* [1] algorithms s.t.

$$\mathcal{K} \times \mathcal{M} \to \mathcal{C}$$
$$\mathcal{K} \times \mathcal{C} \to \mathcal{M}$$

- Correctness property / Consistency equation: for every pair of $(e, d) \in \mathcal{K}$ and for every message $m \in \mathcal{M}$ it must hold that

$$m = D_d(d, (E_e(e, m))$$

- An *advisory* is a entity, not owning the keypair, which is trying to break a cipher, i.e. systematically trying to decode ciphertexts.
- A *secure* cipher is a cipher for which it is provable that no attack in whatsoever form exists on this cipher

According to *Kerckhoff's Principle*, stated by the dutch cryptographer Auguste Kerckhoff in 1883, a secure system should not rely on the secrecy of its components, the only part that should be kept secret is the key alone. Mapped to the definitions above, the sets $\mathcal{M}, \mathcal{C}, \mathcal{K}$, as well as the transformation functions E_e and D_d, must not secret. The only thing that has to be kept private is the keypair (e, d).

FIXME: vergleich security by obscurity

4.3 Kind of Ciphers

Two main kinds of ciphers exist, stream- and blockciphers:

[1] 'runs in polynomial time FIXME: erklären?

Stream Ciphers

For encryption, stream ciphers take arbitrary long messages(from the message space $\mathcal{M}$), and encrypt them to the corresponding ciphertext(out of the ciphertext-space $\mathcal{C}$), by applying one digit of the message to one digit of the key. It is valid to say that a streamcipher is a block cipher with blocklength 1.

- A keystream is a sequence of symbols $e_0, e_1, ..., e_n$, all taken from the keyspace $\mathcal{K}$

The encryption function E_e performs the substitution $c_i = E_e(e_i, m_i)$, producing one encrypted symbol at a time. Analogously, the decryption function inverts this substitution: $m_i = D_d(d_i, c_i)$, with $e_i = d_i$, which means that this kind of cipher is a symmetric one.

Secure Ciphers - The one time pad

As a first example of a theoretically secure cipher the so called 'one time pad' is introduced. This cipher was invented by Gilbert Vernam in 1970, and belongs to the family of stream ciphers.

Shannons Communication Theory

FALSCH: keylength >= message -> secure It is important to state that attacks on any cipher are always possible: the so called 'Exhaustive Attack' will always find the proper key in the long run, just by trying every possible key - no matter what kind of encryption is used. So, this most naive attack will always work, but will not finish in feasible time if a proper key(or more concrete: a proper key-length) is used, turning this bruteforce-attack impracticable. Nevertheless, this statement says that it is principly impossible to design a cipher which can *never* get broken.

Block Ciphers

Here, the cleartext-message is broken into equally sized parts, which are then encrypted block by block. While streamciphers are not parallelizable by nature, there exist methods to speed up en- and decryption by splitting the message respectively ciphertext first as normal, and then process them in parallel[2]. A disadvantage of block ciphers is that it may be necessary to pad the last block to the used block size.

Three main groups of block cipher exists:

- Permutation Blockciphers
- Substitution Blockciphers
- Product Blockciphers
- Feistel Networks

[2]Counter Mode, see 4.3

Permutation Blockciphers

Substitution Blockciphers

Product Blockciphers

Feistel Networks

Stream Ciphers

Block Ciphers

Confidentiality

Cipher Block Chaining - CBC

For encryption, CBC needs as underlying block cipher which is invertible, so a PRP has to be used. As usal, the message has to be broken into blocks, suitable for the block cipher.

$$\begin{aligned} C_0 &= E(k, (M_0 \bigoplus IV)) \\ C_1 &= E(k, (M_1 \bigoplus C_0)) \\ &\dots \\ C_i &= E(k, (M_i \bigoplus C_{i-1})) \end{aligned}$$

To reverse the process, i.e. decrypt the message:

$$\begin{aligned} M_0 &= D(k, C_0) \bigoplus IV \\ M_1 &= D(k, C_1) \bigoplus C_0 \\ &\dots \\ M_i &= D(k, C_i) \bigoplus C_{i-1} \end{aligned}$$

The initialization vector, IV, does not have to be kept private, in fact the receiver of the encrypted message must know this value, either implicitly or explicitply. The first is possible if this IV is some kind of counter or sequence number, which both sender and receiver know. This way, replay attacks can be detected if some kind of MAC is used too, see chapter 4.4. If the IV is chosen by random, or cannot be calculated by the receiver, it **must** be sent along with the message itself as very first block, increasing the overhead, which can be problematic for short messages(for example, consider 1 block messages, consisting of 16 databytes - the IV therefore doubles the size of the data to be sent).

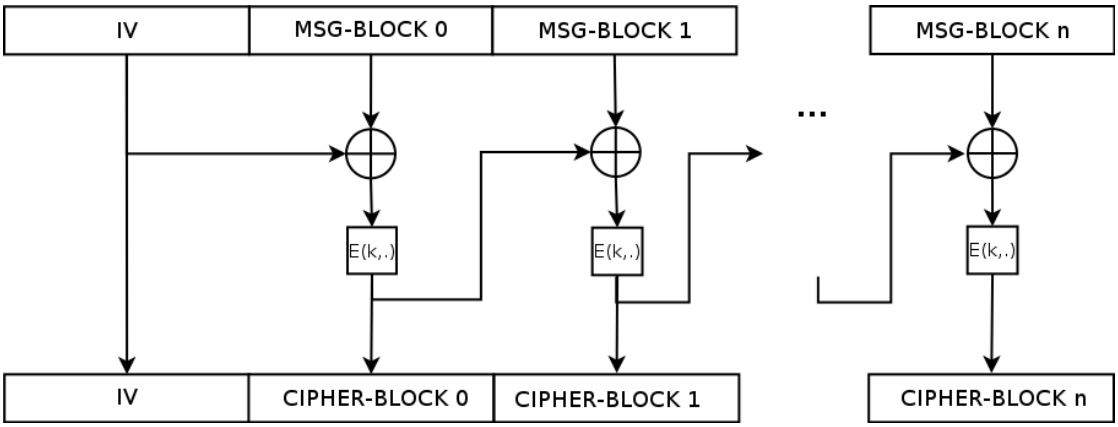


Figure 4.1: Cipher Block Chaining for encrypting messages

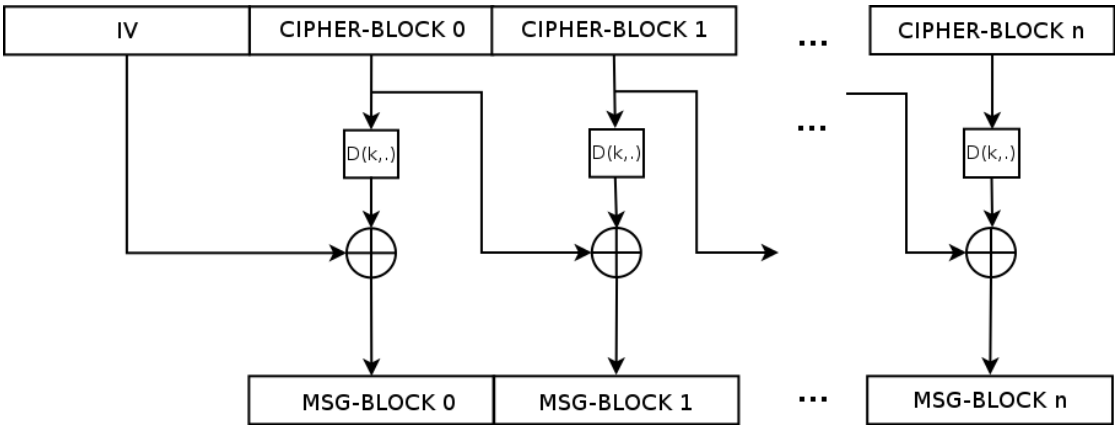


Figure 4.2: Cipher Block Chaining for decrypting messages

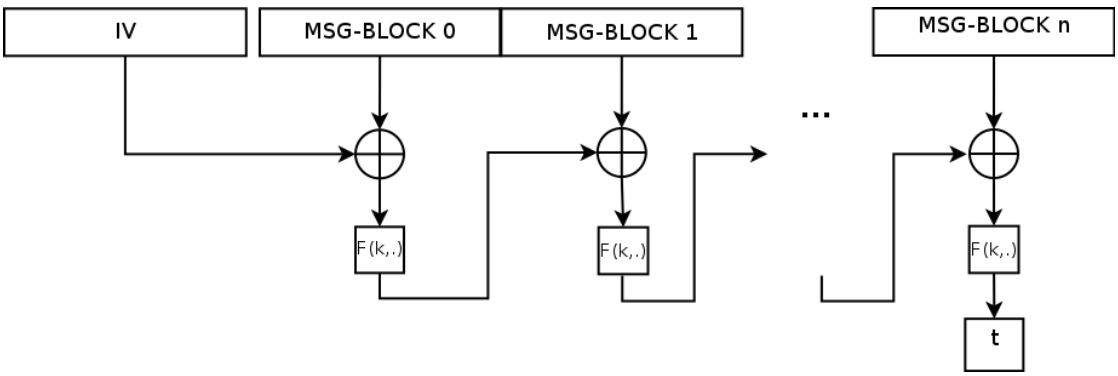


Figure 4.3: Cipher Block Chaining for generating a MAC

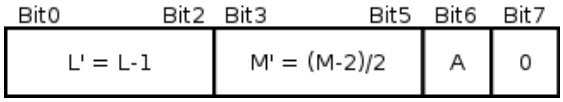


Figure 4.4: Flag Field of CBC IV

Counter Mode - CTR

Authenticity

OCB

Cipher Block Chaining - CBC

4.4 Authenticated Encryption

CCM

CCM[3], short for *Counter with CBC-MAC* combines CBC for authentication and CTR mode for encryption. CBC generates the MAC for the message first, appends this MAC to the cleartext data and afterwards encrypts data + MAC with counter mode, thus using a *MAC-then-Encrypt* scheme. The only supported block size is 128-bit blocks, so it is possible, but not mandatory, to use 128-bit AES as underlying block cipher.
Two application dependent parameters have to be fixed first:

- M: Number of octets in the MAC field. A shorter MAC obviously means less overhead, but it also makes it easier for an adversary to guess the correct value of a MAC, so valid values are $M \in \{4, 6, 8, 10, 12, 14, 16\}$. FIXME: shorter MACs insecure, border=4 ?
- L: Number of octets in the length field. This is a trade-off between the maximum message size and the size of the nonce. Valid values are $2 \leq L \leq 18$. For example, when setting $L = 2$, 2 bytes are reserved for the length field, which means that the biggest message that can be encrypted is of size 64kB. The actual length of the message is filled into the field named 'length(msg)', as shown in figure 4.3.

Both parameters are encoded in the very first byte of the first message block, thus reducing the possible maxium size of the nounce, as shown in figure 4.4. Bit 6 of the length field is set to 1 if additional authenicated data(FIXME) are sent, and bit 7 is reserved and set to 0.

Generating the MAC

As shown in chapter 4.3 in figure4.3, the first message block M_0 is xor'd with a nonce or initialization vector(IV, see figure 4.5), which **must be unique per key**.FIXME The result of the xor operation is then feed to the block-cipher to get the first cipherblock C_0. The encrypted data C_0 gets xor'd with the next message block M_1, and this result becomes the input for the block cipher, and so on, iterating over all n message blocks to determine the tag t:

[3] http://tools.ietf.org/html/rfc3610

Figure 4.5: IV for CBC MAC

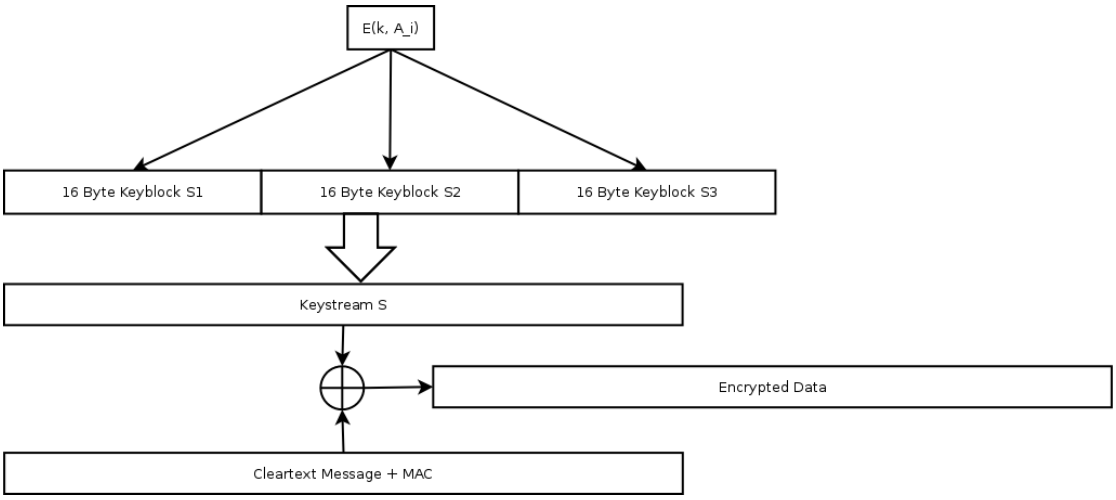


Figure 4.6: CTR Encryption

$$
\begin{aligned}
C_0 &= F(k, M_0 \bigoplus IV) \\
C_1 &= F(k, M_1 \bigoplus C_1) \\
&\dots \\
C_n &= F(k, M_n \bigoplus C_{(n-1)})
\end{aligned}
$$

The resulting tag t can be truncated, corresponding to the chosen MAC size M:

$$t = C_n[M : 0], \text{ with } M \in \{4, 6, 8, 10, 12, 14, 16\}$$

which means that the tag t consists of the least significant M bytes of the output of the last encryption block.

Encrypting Data and MAC

Counter-mode is used for encrypting the actual payload and the concatenated, CBC mode generated MAC. Thus, authenticated encryption is achieved in a manner also called 'mac-then-encrypt'. While authenticated encryption modes implementing this ordering(generate mac first, then encrypt data and mac) *may* be vulnerable to padding oracle attacks(FIXME), counter mode effectively avoids these simply because there is no padding needed, as will be shown.

Counter mode implements a weaker form of the one time pad by generating a keystream of sufficient length, and then xoring the keystream with the data itself, as shown in figure 4.6.

First, keyblocks with 16 byte length each are generated by encrypting the nounce, a flag and a counter with the key. These keyblocks are then concatenated and trimmed to the proper

length(=length of the message to encrypt). This obtained keystream is then bitwise xored with the cleartextmessage(which consists of the data and the MAC), yielding the final encryption.

Decryption and Authenticity Check

Attacks on CCM

FIXME: meet in the middle attack, siehe rfc 3610

4.5 Public Key Cryptography

Public Key Cryptography solves the problem of establishing a secure channel by using an unsecured one. Here sender and recipients use two different keys: one for encryption, called *public key*, the other for decryption, called *private key*. This key pair belongs together, hence this scheme is also called *asymmetric* encryption. A key requirement is that it must be hard to derive the decryption key from the encryption key. This behavior is achieved by some kind of public known one-way function where it is computationally easy to calculate the result of $f(x) = y$, but only given y, it is computationally - in the domain of processing power and/or memory - hard to reverse this function to get x, although the reverse function may exist in mathematical sense. This is even a desired property. Otherwise it may facilitate to find the argument that led to the output, i.e. take the constant function, where it is trivial to find the argument. By that fact, the encryption or public key can be published in some sort of dictionary without compromising the private key. An entity wanting to send an encrypted message to a receiver can then look up the receiver's public key, encrypt the message and send the resulting ciphertext to the recipient, who then can decrypt the message. It is remarkable that any algorithm establishing public keys must authenticate it's participants, or it will be vulnerable to man-in-the-middle attacks.

Discrete Logarithm Systems

Whitfield Diffie and Martin Hellman were the first who proposed a way to solve the problem for key-exchange by introducing the concept of a public-key cryptography when they published their paper *New Directions in Cryptography* back in 1976. The security of this concept is based on the hardness of the *Discrete Logarithm Problem*.

With the original Diffie-Hellman algorithm, 2 entities - A and B - use exponentiation over finite fields to agree on a shared secret, which then can be used parametrize a block or stream cipher. The first step for booth entities is to agree on the set of parameters $\{p, q, g\}$, where p is a large prime, q is a prime divisor of $p - 1$, and g is a generator of the cyclic group $Z_p{}^*$ in the range $[1, p - 1]$. These parameters are not secret and can thus be sent over an unsecured channel. Additionally, each entity randomly chooses an integer x from the interval $[1, q - 1]$, and calculates the value $y = g^x \pmod p$. x is the private key, y, which is computationally easy to calculate, is the public key. A sends its public key $y_A \equiv g^{x_A} \pmod p$ to B, and B its public key $y_B \equiv g^{x_B} \pmod p$ to A. Due to the characteristics of exponentiation, A and B can now easily derive the shared secret by using it's counterpart's public key and raising it to the power of it's own private key in the domain of $Z_p{}^*$:

$k_B \equiv {y_A}^{x_B} \equiv g^{x_A x_B} \equiv g^{x_A * x_B} \pmod{p} = k_A \equiv {y_B}^{x_A} \equiv g^{x_B x_A} \equiv g^{x_B * x_A} \pmod{p}$

An eavesdropper that intercepts the initial sent paramter set $\{p, g, q\}$ and the public keys y_A and y_B and that wants to calculate the shared secret $k_A = K_B$ must therefore the Discrete Logarithm Problem. FIXME: security analysis of DLP

Diffie-Hellman based on Elliptic Curves

RSA

4.6 Random Number Generators

Quality / Measurement

Hardware based

usb stick: www.entropykey.co.uk

4.7 Attacks on Ciphers

Passive Attacks

timing attacks - constant time computation

Active Attacks

CHAPTER 5

Security Concept

5.1 Basic Assumptions

One of the main purposes of this work is to establish secure knx communication in a transparent way, so a device outside of this network, unaware of the secured knx network, should be able to deliver through and receive messages from such a secured network without any prerequisites. Every device with one connection to an unsecured knx network and at least one connection to a secured knx network, running the master daemon, will work as a security gateway. Thus, the presence of at least 2 of these security gateways connected to each other by one or more secure lines will constitute a secured knx area, bridging between areas with low security levels, as shown in figure 5.1.

The basic tasks of such security gateways consist of:

- establishing keys with its communication partners within the secured knx network(the security gateways)
- maintaining some kind of synchronization token between all security gateways
- encrypting and authenticating all messages which are received on the unsecured line, and delivering them to the proper security gateways which act as border device for the given group address, using booth secure lines, to achieve redundancy
- checking all messages which are received on the secured lines for integrity and authenticity, removing duplicates, unwrapping and delivering them to the unsecured area

As stated in chapter 2, 3 different possibilities for communication within a KNX network are possible: point to point, multicast and broadcast. To introduce as little additional traffic into the network as possible, a sound concept for translating of clear- to secure-KNX address(and vice versa) has to be defined. While in principle it would be possible to use the communication modes in a transparent way(for example, point-to-point in unsecured knx translates to point-to-point in

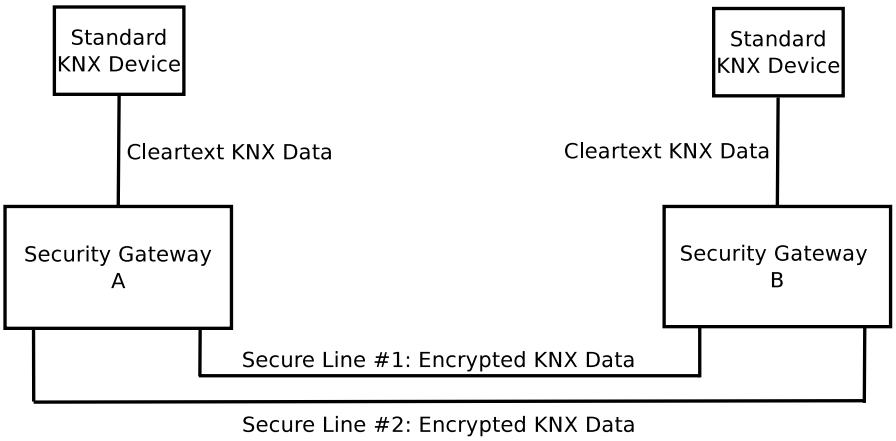


Figure 5.1: Secure Area

secured knx, and vice versa, and so on), this approach leads to some serious problems, rendering this solution impracticable: due to the topology of KNX, it is impossible to know a priori the exact physical location of a device(i.e., its individual address). Additionally, every device can be member of an arbitrary number of group addresses(bounded by the maximum number of group addresses), which also is not known in advance. Group-membership is also subject to change and therefore worsens the situation. Finally, devices can leave or join the network at every moment by powering the device up or down. Therefore, a device which receives a message on its unsecured knx line, examining the destination address, simply does not know which device(s), if any, will be the gateway(s) responsible for delivering this datagram one hop toward its final destination, regardless if the destination address is a group- or an individual address.

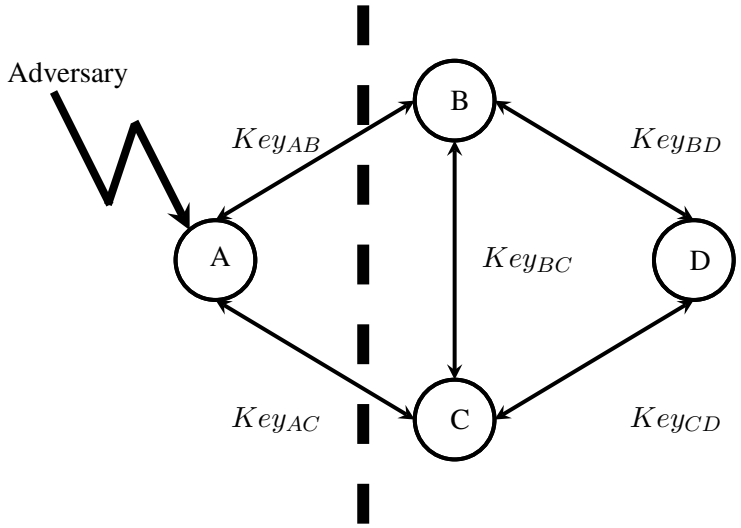
A straightforward solution to this problem would be to wrap every datagram which enters the secured knx network via a security gateway into a new, properly secured broadcast datagram, and delivering this new package to the secured knx network. Then, the package would be available to all other security gateways, which will unwrap it and forward the resulting inner datagram to its unsecured knx line. If the destination address(group or individual) of the actual payload is assigned to a device connected to the unsecured knx network, the device holding this individual- or group-address will recognize it and the package will reach its destination. Otherwise it will simply be discarded.

A serious constraint rising from this broadcast approach is that a single, global network key must be used, because every security gateway must be able to decrypt and check every package which it receives on it's secured lines, even if it does not serve as gateway to the wanted group address. The key of course can be renegotiated among the security gateways at every time, but this approach is considered unsafe because an attacker can target *any* of the security gateways constituting the secure network. An adversary breaking one single device gains access to the network traffic of all devices. This could be achieved by physical access to any of the security gateways, for example by reading out the memory of the device, and thus obtaining the globally used network key. This way, the network traffic can be decrypted by the adversary as long as no new key is renegotiated. Another problem is that multi-party key negotiation is a costly task if a

public-private key scheme is to be used: as shown in figures 5.2 and 5.3, a lot of messages have to be exchanged before actual an encryption can be done.

To encounter this problems, different keys must be used. This way it is also possible to achieve different security levels, depending on the function a particular unsecured knx device fulfills. It would be possible, for example, to distinguish between 'normal' gateways and 'hardened' gateways which are specially guarded against physical access, for example by applying physical intrusion detection. Thus, the risk of breaking the whole system is reduced, because breaking a device in one security level does not affect the security of the devices with other security levels. So, for breaking all n security levels of a system, at least n devices, all belonging to different levels must be broken. As a motivating example, imagine a setup which consists of window controls in an upper floor, and door controls in the base level. Obviously, the security constraints for the latter one would be higher. By using normal devices for window control, and hardened devices for door control, a security firewall can be deployed, thus containing the damage an adversary can do to the whole system.

Figure 5.1 shows the logical connections within an KNX network with different security levels. An attack of node A can only compromise keys known to the device, thus effectively separating communication between the nodes B, C, and D from the attacker.



But, as stated above, to be able to use different keys, every security gateway has to know how to reach a given address, so that the data can be encrypted exclusively for the responsible gateway. The solution to this problem is to maintain some kind of routing table, mapping group and individual addresses of unsecured knx devices to individual addresses of responsible security gateways. Additionally, this table must hold the key that will be used for encryption. Such a routing table can be built statically at setup time, with the obvious disadvantage that the exact topology of the to be applied network has to be known in advance, thus reducing the flexibility. Here, every security gateway holds a static table which consists of mappings between individual- or group addresses of unsecured knx devices and individual addresses of security gateways at the border between the secured and the unsecured knx network, as well as all keys

used for the particular security level the gateway belongs to. This table would be generated once, after the topology of the network has been fixed, must be equipped with the proper keys and can then be copied to the security gateways constituting the secured knx area. New security gateways can be deployed as long as they only introduce sending unsecured knx devices, whose recipients are already mapped, known group addresses. A new group address, introduced by a newly installed device behind an already existing security gateway, will not be reachable, simply because the routing information and the encryption key is not available. Another disadvantage is that the deployment of new security gateways, connecting devices with new or already known group addresses, is pointless as the individual address of the new gateway - which of course must be unique - is unknown to the existing setup, thus making the new unsecured knx devices unreachable.

To tackle this problem, another approach would be to build this mapping table dynamically. Therefore, every security gateway must periodically poll on it's unsecured lines for KNX devices(FIXME: HOW? analog zu ETS group address polling), thus populating a list of reachable knx devices. Whenever a device wants to send data to a group address, it has to do a lookup first to obtain the individual addresses of the responsible security gateways: the lookup must contain the wanted group address, as well as the senders public key. Every gateway which finds the wanted group address in its group list must reply with an according message to the requester, thus announcing that it is responsible for delivering data to the wanted group address, and must also publish it's own public key, thus allowing pairwise end-to-end encryption. The original requester must wait for a short time for replies, possibly retransmitting the request in case of no responses, and can then transmit the encrypted package to all responsible gateways, if any, one at a time. This procedure requires no a priori knowledge of the network topology, so security gateways can be added to the network as well as unsecured knx devices behind new or existing gateways at any time. This flexibility of course has to be purchased with increased complexity as well as additional traffic induced into the network.

As a middle course it would be viable to generate the reachable group address list whenever a new security gateway is added to the network, and handle discovery of this group addresses as described above. This makes it possible to deploy new security gateways with connected unsecured devices, thus allowing a comprise between flexibility and complexity.

Security Related Architectural Overview

To provide authenticity, all datagrams passing the secured knx network must contain a MAC to prevent modification of them(i.e. to guard against active adversaries). This mac must be combined with a counter value to avoid replay attacks. The counter must be strictly monotonically increasing and must not overflow. The counter can be seen as initialization vector that prevents the mapping of same cleartext messages to same ciphertext messages under the same encryption key. To guard against passive adversaries, i.e. eavesdropping, all datagrams carrying knx application data must be encrypted. These are all datagrams coming from outside of the secured area, originating from an unsecured knx device. As explained above, these packages will be encrypted end-to-end, with unique asymmetric keys between each two communication partners. Additionally, all discovery messages generated by security gateways will be encrypted too. Although these datagrams don't contain knx data per se, they allow a listening adversary

to learn the topology of the network, knowledge which can be valuable for developing an attack strategy, as well as generating meta data. For example, if an attacker learns that a particular security gateway is responsible for only one group address, and she further gets knowledge that this group address is responsible for switching a light(i.e. by visual observation), she afterwards may be able to derive a personal profile just by seeing packages for this group address, although the datagrams are encrypted. If the discovery messages are encrypted too the adversary doesn't know how many group addresses are behind the gateway, and it will be harder to derive personal profiles.

Redundancy Related Architectural Overview

To achieve a higher level of availability, all components that are needed to provide a specific service must exist multiple times. Whenever a knx package is generated by a device on an unsecured line(called client), the connected security gateway will read, duplicate and encapsulate it into another knx frame and then send over booth lines. If booth lines are available, i.e. there is, for example, no shortcut, a receiving security gateway will receive 2 different knx frames encapsulating the same payload, which itself is the knx frame generated by the knx device in the first place. One message must be discarded to avoid duplicates. This is achieved with the counter that guards against replay attacks: every time a security gateway encapsulates a datagram and tries to send it over booth lines, 2 counters are incremented(one for every lines). Each of these 2 counters belong to distinct source-destination tuples, and are also included in the package and are used as input when generating the MAC.

The need for such a counter implies that some kind of synchronization service must be available. A security gateway which has lost it's synchronization with the rest of the KNX network, or which is powered up and wants to join the network must use this service to obtain the actual value of the counter. The counter value needs not to be secret, but must be authenticated. Otherwise, a malicious device can fool a synchronizing device into using false counter values and thus trigger an denial of service attack onto this device.

Operational Constrains

The introduction of encapsulating security gateways implicates that some timing constraints, defined by KNX, cannot be met:

- Acknowledge frames, as defined in KNX and introduced in chapter 2, cannot be guaranteed to be delivered within the specified deadlines: whenever a new KNX datagram is generated by a client, at first the discovery phase has to occur. Only after that the to-acknowledged frame is sent. So there are multiple delays introduced, stalling the delivery: the first delay is caused by sending of the discovery package. After that, a second delay occurs because the security gateway must wait for the discovery response(s), possibly retransmitting the discovery request in case of a timeout. After receiving discovery responses, the third delay is caused by sending the actual, encapsulated client package to the responsible security gateway(s), which then must check the datagram, unpack it and forward it on it's unsecured line. Only after that, all addressed, unsecured clients would be

able to acknowledge the received frames to their local security gateways, which must forwarded the acknowledgement frame to the origin security gateway, causing another delay. Finally, the acknowledgement frame must be forwarded to the sender of the origin data frame, causing another delay. These delays will always occur, and most of them cannot be restricted, thus destroying the tight timing constraints for acknowledgement frames, as defined by the KNX standard. This will most likely result in multiple retransmissions of the same KNX packages by the client because the client's timer will generate a timeout. The only way to solve this is to immediately acknowledge a client frame by the security gateway that it is connected to. On the receiving side, the client will generate an acknowledge frame, which must be discarded by it's security gateway.

- Similar arguments avoid the processing of Poll-Data Frames. Here, event more stringent timing constraints are to be met, see chapter 2.

Operation states FIXME

synchronization
    joininig
    discovery
    data

Key Management

The previous statements imply that 3 different kind of keys must be used:

- First, a key known to all security gateways is used. As already stated, this key must be copied to every device at setup time so that it is known to all devices. This is a pre-shared key, named k_{psk}, used for symmetric encryption. This key serves for 2 different purposes: First, it is used as authentication token to synchronize new devices which want to join the network, as well for devices that have lost their synchronization(i.e. that have been unavailable for some time). Additionally, this key k_{psk} is used to encrypt another symmetric key k_{global}, which every device must obtain in the joining phase to be able to take part in the discovery procedures.

- k_{global} is used to authenticate and encrypt locally generated and decrypt received discovery requests, as well as to authenticate and encrypt locally generated discovery responses and decrypt received ones. This discovery service datagrams securely transport the third type of keys:

- Asymmetric keys are used for end-to-end encryption of the actual data packages between 2 security gateways.

Discovery of Group Addresses

To keep the mapping between group addresses and individual addresses of responsible gateways up to date, a discovery service is defined. Because an important information this mapping holds is the

5.2 Key Derivation

Key derivation is the process of establishing parameters for secure communication between 2 or more communication partners, most importantly, a shared key which is used to encrypt and/or authenticate data, but also parameters like key length and which encryption and authentication primitives to use. Because symmetric key encryption outperforms its asymmetric counterparts(FIXME: benchmarks symmetric vs. asymmetric @pi) in regards of performance, a hybrid approach is taken. In the very beginning of key negotiation, no secure channel is available, so some kind of already known authentication token must be used. This can be a key, known to all devices, called a pre-shared-key.

so an asymmetric encryption scheme is used. The asymmetric keys are used to derive the actual session key, which is volatile and can be re-negotiated at any time.

While it would be possible to use a centralized concept, no trusted on-line party(key server) is used in this work. This de-centralized setup is used to simplify the setup. A centralized approach would need fall-back key-servers which inherit the task of generating and distributing keys and parameters in case of a master key server failure. This key server would nevertheless need some singular authenticating property which must be known to all possibly joining devices, so a different approach is taken here: a so called *pre-shared key* k_{psk} is used for authentication.

This key serves as entry point into the secured network, authenticating messages between new joining and already joined devices. While it would be possible to also encrypt messages at this stage, it is to be noted that here, due to the characteristics of asymmetric encryption, it is sufficient to authenticate the messages because no secret parameters will be transmitted in this phase. A joining device is a device which is booted up and gets connected to the existing secure knx network, and which wants to become part of this network. The actual result of 'joining' is to obtain all parameters which are necessary for encrypted and/or authenticated communication with other devices in the network.

The pre-shared key must be kept secret, and has to be known to all devices which want to join the secured knx network. It is used to prove the identity of the new device to the already joined, other devices. Because this setup key is used as single security token, it is important to note that it is **impossible** to distinguish between a regular and a malicous device which booth have knowledge of the pre-shared key.

Diffie - Hellman

If possible, achieve *perfect forward secrecy* by using Diffi-Hellman. On the other hand, a single network key, known to **all** devices in the secure KNX network, has to be used. This constraint rises from the fact that within the secured network, it is not known **where**, and even stronger, **if**

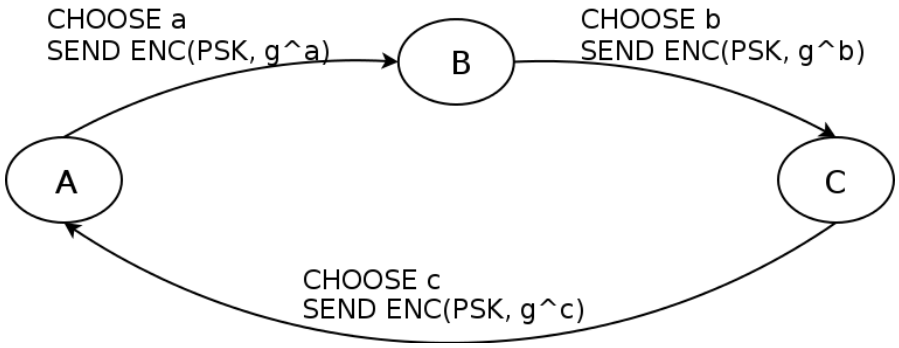


Figure 5.2: DH Round 1

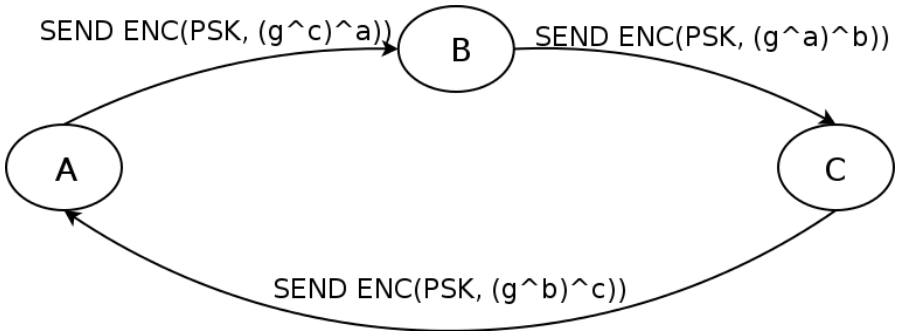


Figure 5.3: DH Round 2

the recipient of the to be secured message is connected to a device at the border of the secured/unsecured network. Of course, it would be possible to encrypt the origin, unsecured message on a peer-to-peer basis, and send this message to **all** devices within the secured network, but this obviously would flood the message, adding additional busload for every new device within the network, so this way is considered not feasible.

One Secret for all devices

parties A, B, C, with one known generator g

1. first iteration, see 5.2:

   A: chooses private key a, calcs $x_a = g^a$, send to B ENC(PKS, x_a)
   B: chooses private key b, calcs $x_b = g^b$, send to C ENC(PSK, x_b)
   C: chooses private key c, calcs $x_c = g^c$, send to A ENC(PSK, x_c)

2. second iteration, see 5.3

   A: calc $(g^c)^a$, send to B
   B: calc $(g^a)^b$, send to C
   C: calc $(g^b)^c$, send to A

3. third iteration: calulcate shared secret

   A: calc $((g^b)^c)^a$
   B: calc $((g^c)^a)^b$
   c: calc $((g^a)^b)^c$

$((g^a)^b)^c = g^{a*b*c} = KEY$

This key is used to further derive 2 new keys: 1 MAC key, 1 Encryption key and can be generalized for n parties.

- Pro: one shared key for all parties
- Contra: for every new joining party, the whole key derivation rounds have to be done
- Contra: if one node is not reachable temporary or leaves network and another party wants to join, a new key is derived. if temporary unavailable node is reachable or again, new key has to be derived too.

Secret for all pairs of devices

1. one device A present: A: choose private exponent a
2. seconde devices B wants to join, see 5.4:
   B: choose private exponent b, send E(PSK, "Hello" + g^b)
   A tries to decrypt the Message, if it can retrieve the string Hello the message originates from an allowed sender and answers with ENC(PSK, "Welcome" + g^a)
   A and B have now share the common secret $s = (g^a)^b = g^{a*b}$
3. if new device C want to join, see 5.5
   choose private exponent c, send ENC(PSK, "Hello" + g^c)
   A: answers with ENC(PSK, g^a)
   B: answers with ENC(PSK, g^b)
4. for every new device such a 2 round iteration has to occur

- Pro: fewer messages for key derivation
- Pro: devices can leave network or become unavailable without disturbing key derivation of other nodes
- Contra: $\frac{n*(n-1)}{2}$ keys for n devices

As shown in the beginning of the chapter, peer-to-peer keys cannot be used due to the broadcast nature of the knx network.

Figure 5.4: DH 2 Parties

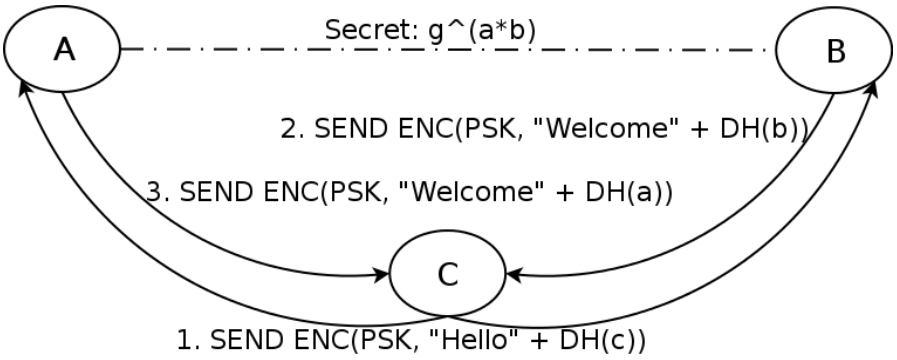


Figure 5.5: DH 3 Parties

Using the Factory Key

Using a fixed String as Authentication Token

This first simple protocol will work only for passive adversaries. If active adversaries are present, it is vulnerable to replay attacks, although this kind of attack would bring no benefit for the adversary because she cannot get the actual sessionkey, because of the lacking of the preshared key.

1. A wants to 'join' a network which is unpopulated at this time:
   - sends $c = E(k_{psk}, "Hello")$
   - timeout, no response due to the 'empty' network
   - A randomly chooses the session key k_s and resets the seqeuence number
2. B wants to join the network
   - B sends $c = E(k_{psk}, "Hello")$
   - A decrypts ciphertext, **iff** decryption succeedes(i.e. received cyphertext decrypts to $"Hello"$): A waits for a short, randomly choosen time and sends $c = E(k_{psk}, k_s)$

- A: if decryption fails(i.e. c **does not** decrypt to "$Hello$" this means that an advisory is trying to enter the secured network and drops the message
- B decrypts $k_s = D(k_{psk}, c)$

Challenge - Response

1. A wants to 'join'a network which is unpopulated at this time:
   - sends unencrypted 'Hello' message
   - timeout, no response due to the 'empty' network
   - A randomly chooses the session key k_s and resets the seqeuence number
2. B wants to join the network
   - B sends unencrypted 'Hello' message
   - A chooses a random number n and sends n unencrypted to B
   - B(legimitate user or adversary) can always encrypt the number and reply the value to A
   - A can compare the sent encryption of n by itself generating the encrypted value under k_{psk}. Iff the values match, A replies with $c = E(k_{pks}, k_s)$. Otherwise, it drops the message, considering B as an adversary.

Instead of a random number n, it would also be possible to use a timestamp of sufficient granularity, which would also provide data freshness. A drawback is that the clock of a joining party must be syncronized.

High Level Cryptography Library

OpenSSL

- install libssl, libssl-dev

Crypto++

- install libcrypto++9

CHAPTER 6

Implementation

6.1 Setup of the Base System

The base system consists of a standard raspbian operating system, the EIBD daemon, shared libraries which are used by EIBD and the master daemon. The operating system is based on the Debian project, with the kernel, libraries and binaries ported to the ARM platform, so it is possible to benefit from using a full-scale operating system, e.g. by using the comfortable packet manager called *aptitude* provided by Debian. A short introduction to the most important commands is given below as they are needed.

Raspbian

To obtain a running system for deploying the secure KNX daemon, a prebuilt Debian image is used, which can be ownloaded from the raspberry homepage:

`http://downloads.raspberrypi.org/raspbian_latest.torrent`

The image must be unzipped and copied to a suitable memorycard. First-generation raspberries(model 'A') have SD slots, while all later models come with micro-SD slots. To copy the basic operating system to the memorycard, the linux commandline tool 'dd' can be used. To find the correct device to write the image to, the following command can be used:

```
1  # tail -f /var/log/kern.log
```

After inserting the memorycard into a cardreader, look for output like that:

```
1  [1004111.533698] sdb: detected capacity change from 7909408768 to 0
2  [1004114.055840] sd 6:0:0:0: [sdb] 15448064 512-byte logical blocks: ...
```

Here, the proper device to use is the device /dev/sdb. **Pay attention to use the correct device in the following command - this device will be overwritten**:

```
1  # dd if=<Path to Image> of=<Device to overwrite>
```

After the write command has finished, the memory card is ready to use. For first time setup, a display must be connected via HDMI. Powering up the raspberry opens a ncurses configuration dialogue. First thing to do is to resize the root partition to maximum size and set a password for the administrative account. Optionally, different options like keyboard layout can be set. To be able to operate the raspberry without external display, it is necessary to start the secure shell (SSH) server under *Advanced Options* and assign a fixed ip to the host by editing the file */etc/network/interfaces*, as shown in example A.1. This way it is possible to connect to the raspberry with a SSH client. For password less logins, create an unpriviliged user and a SSH public/private key pair for that user by executing these commands on the raspberry pi:

```
# groupadd <usergroup>
# useradd -g <usergroup> -m <username>
# su <username>
# ssh-keygen
```

The program generates the user and the correspoding key pair and saves public and private key in the subdirectory /.ssh/ on the actual host. When asked for a passphrase, it is possible to use a password-less keypair, an option that should only be used in restriced areas. To actually use the keypair for logging into the raspberry pi, the public key must be saved in the file */.ssh/authorized_keys*. Additionally, the private key must be copied to every host from that SSH connections to the raspberry pi want to be opened. After that, it is possible to load the private key into memory with the command *ssh-add* and to connect to the host without a password:

```
# ssh-add // only necessary when non-empty passwort is used for keypair
# ssh <username>@<host-ip|host-dns-name>
```

It is also advisable to update the operating system at this time by running the following commands as user root:

```
# apt-get update
# apt-get install
```

This will install the latest package versions of all installed packages. New software can be installed from the command line with these commands:

```
# apt-cache search <pattern> // print a list matching packages for <pattern>
# apt-get install <packagename>
```

EIBD

The maintainer of EIBD only provides binary packages for the i386 architecture, so the daemon and its prerequisites must be built from source to get suitable binaries and shared libraries for the ARM environment. Building software under GNU Linux or *nix from source always follows this scheme:

1. Downloading and extracting the source code

2. If possible, comparing the developer supplied hash code with the hash code of the downloaded source files with *sha256* or one of its variants to verify that no modified software has been downloaded.

3. Optionally, apply patches to the source code(not necessary here).

4. Set the make-options by calling *./configure* <*options*>, overriding default compilation options by setting the corresponding command line parameters. *./configure –help* should print a list of valid options.

5. Compiling the source code by calling *make*.

6. Copying the generated binaries and shared libraries into their correct place by calling *make install*. This last step must always be executed as user root because the generated files will be copied into system directories which are not writeable by unprivileged users.

EIBD and the needed library *pthsem* are available from these locations:
`https://www.auto.tuwien.ac.at/~mkoegler/pth/pthsem_2.0.8.tar.gz`
`http://sourceforge.net/projects/bcusdk/`

After copying the archives to the raspberry, they must be unpacked and compiled. First the pthsem shared library, which offers user mode multi threading with semaphores, must be compiled because it is used by EIBD.

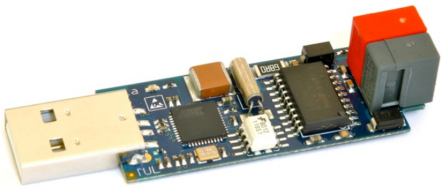
```
1  # tar -xvzf pthsem-2.0.8.tar.gz
2  # cd pthsem-2.0.8
3  # ./configure
4  # make
5  # make install // must be executed as root
```

This will, among other things, generate the shared library *libpthsem.so.20* in the directory */usr/local/lib*. */usr/local* is by convention the destination where self compiled software should reside. Now that pthsem is available, which is a dependence of the EIBD daemon, EIBD itself is ready for compilation:

```
1  # tar -xvzf bcusdk-0.0.5.tar.gz
2  # cd bcdusk-0.0.5
3  # ./configure --without-pth-test --enable-onlyeibd --enable-tpuarts
4  # make
5  # make install // must be executed as root
```

These steps generate the binary *eibd* and lots of helper programs in the directories */usr/local/bin*, and the shared object */usr/local/lib/libeibclient.so.0* that provides the european installation bus daemon (EIBD) application programming interface (API) and therefore is needed to be linked to the master daemon.

Figure 6.1: Busware KNX-USB coupler



Revision control

The source of the master daemon is managed by GIT. GIT is a decentralized revision-control system and is available under Debian/Raspbian after installing the package 'git'. The command 6.1 fetches the latest version and creates a directory called 'knxSec' which contains all the needed source files, a proper makefile A.3 for the project, as well as all other needed files.

```
1  # git clone git@github.com:hglanzer/knxSec.git
```

Busware universal serial bus (USB) couplers

To make the KNX TP1 bus accessible, i.e. to write datagrams to and receive datagrams from the bus, USB dongles as shown in figure 6.1 from the company *Busware* are used. Depending on the revision, the bus couplers creates a new device which is used as uniform resource locator (URL) by the EIBD. The coupler will be accessible by */dev/ACMx*, where x is the number of the device. It may be necessary to flash the bus couplers with the correct firmware first. The easiest way to check this is to use command 6.1 and look for output similar to listing 6.1 when plugging the coupler into an USB slot.

```
1 ... usb 1-1.2: new full-speed USB device number 19 using ehci_hcd
2 ... usb 1-1.2: New USB device found, idVendor=03eb, idProduct=204b
3 ... usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=220
4 ... usb 1-1.2: Product: TPUART
5 ... usb 1-1.2: Manufacturer: busware.de
6 ... usb 1-1.2: SerialNumber: 7543034373135130C140
7 ... cdc_acm 1-1.2:1.0: ttyACM0: USB ACM device
```

If no such line like 7 appears, the correct firmware is available as file *firmware/TPUART-transparent.hex* inside the git project. To actually flash the coupler, the programming button on the bottom of the device must be kept pressed while connecting it to an USB slot. Afterwards, the commands shown in 6.1 must be executed.

```
1  # apt-get install dfu-programmer
2  # dfu-programmer atmega32u4 erase
3  # dfu-programmer atmega32u4 flash TPUARTtransparent.hex
4  # dfu-programmer atmega32u4 reset
```

Test setup

The test environment consists of 2 raspberry pis, as shown in figure 5.1.

6.2 Master daemon

Bibliography

[1] KNX Association. “KNX Applications”. URL: http://www.knx.org/fileadmin/downloads/08%20-%20KNX%20Flyers/KNX%20Solutions/KNX_Solutions_English.pdf.

[2] KNX Association. “System Specifications, Overview”. URL: http://www.knx.org/knx-en/knx/technology/specifications/index.php.

[3] N. Borisov, I. Goldberg, and E. Brewer. “Off-the-record Communication, or, Why Not to Use PGP”. In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. Washington DC, USA: ACM, 2004, pp. 77–84.

[4] S. Cavalieri and G. Cutuli. “Implementing encryption and authentication in KNX using Diffie-Hellman and AES algorithms”. In: *Industrial Electronics, 2009. IECON ’09. 35th Annual Conference of IEEE*. 2009, pp. 2459–2464.

[5] S. Cavalieri, G. Cutuli, and M. Malgeri. “A study on security mechanisms in KNX-based home/building automation networks”. In: *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. 2010, pp. 1–4.

[6] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. RFC 2246. RFC Editor, 1999, pp. 1–80. URL: https://www.ietf.org/rfc/rfc2246.txt.

[7] W. Diffie and M.E. Hellman. “New directions in cryptography”. In: *Information Theory, IEEE Transactions on* 22.6 (1976), pp. 644–654.

[8] I. Goldberg et al. “Multi-party Off-the-record Messaging”. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS ’09. Chicago, Illinois, USA: ACM, 2009, pp. 358–368.

[9] W Granzer, G. Neugschwandtner, and W. Kastner. “EIBsec: A Security Extension to KNX/EIB”. In: *Konnex Scientific Conference*. Nov. 2006.

[10] L. Hong, E. Y. Vasserman, and N. Hopper. “Improved Group Off-the-record Messaging”. In: *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. WPES ’13. Berlin, Germany: ACM, 2013, pp. 249–254.

[11] S. Kent and K. Seo. *Security Architecture for the Internet Protocol*. RFC 4301. RFC Editor, 2005, pp. 1–101. URL: http://tools.ietf.org/html/rfc4301.

[12] R. L. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-key Cryptosystems”. In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126.

[13] S. Vaudenay. “Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS”. In: *Proceedings of In Advances in Cryptology*. Springer-Verlag, 2002, pp. 534–546.

Glossary

API application programming interface. 35

EIBD european installation bus daemon. 35, 36

SSH secure shell. 34

URL uniform resource locator. 36

USB universal serial bus. 36

APPENDIX A

Code snippets and configuration files

Listing A.1: Raspbian configuration for static ip address

```
# device: eth0
  auto eth0
  iface eth0 inet static
  address 192.168.0.2
  broadcast 192.168.0.255
  netmask 255.255.255.0
  gateway 192.168.0.1
```

Listing A.2: Raspbian configuration for dynamic ip address

```
# device: eth0
  iface eth0 inet dhcp
```

Listing A.3: Makefile for the master daemon

```
CFLAGS=-Wall
LIBS=-leibclient
#LIBS=-lgmp -lcrypto -llibeibclient

all: clean update
	gcc $(CFLAGS) $(LIBS) master.c sec.c cls.c -o master -pthread
	#gcc $(CFLAGS) master.c sec.c cls.c -o master /usr/lib/libeibclient.so.0
	    -pthread

debug: clean
	gcc $(CFLAGS) master.c sec.c cls.c -o master /usr/lib/libeibclient.so.0
	    -pthread -DDEBUG
	./master

clean:
```

```
14          clear
15          rm -rf *.o
16          rm -f master
17
18  run: all
19          ./master
20
21  update:
22          git commit -a --allow-empty
23          git pull
24          git push
```