FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

KNX for Safety Critical Environments

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

B.Sc. 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)

KNX for Safety Critical Environments

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

B.Sc. 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

B.Sc. 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. Atten-

dance 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

State of the art

2.1 Security

The basic building stones of information security are confidentiality, integrity and availability, also called the *CIA - triad* [1]. **Confidentiality** is used to protect sensitive information from eavesdroppers who are not allowed to get knowledge of that information. **Integrity** ensures that some kind of information can not be altered by third-parties, or that such a modification can be detected by the receiver of the information, and also includes information non-repudiation. **Availability** ensures that information, which is needed by an entity to provide some service, is accessible. All three properties go hand-in-hand with each other because a successful attack on one property may allow attacking another one. For example, if a confidentiality attack against a computer system responsible for money transfers can be conducted to steal a password used for controlling this system , an attacker can subsequently render the system unusable, therefore compromising availability. On the other hand, the attacker could also try to remain undetected and change booking orders, thus mounting an attack against the integrity of the system. Thus, these 3 basic concepts are interleaved, and building a system which honors only parts of them will most likely lead to an insecure system.

Additionally to the CIA - triad, sometimes two more concepts are used in the security field: **Authenticity** is tied to integrity and ensures the property of being genuine, while **Accountability** allows to link actions performed on a system uniquely to the entity responsible for them.

2.2 Cryptography

The tool to achieve information security is cryptography. Cryptography [1] is the science of encrypting information. 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 table for prominent events is presented:

One of the oldest witnesses for cryptography are hieroglyphs used in Egypt about 2000 B.C., forming the predecessor of a simple substitution cipher. 500 B.C., the "skytale" was used by greek and spartan military leaders, performing a transposition cipher. Another classical example was the "Caesar Cipher", used by its inventor about 100 B.C. to hide information by replacing every letter of the alphabet by a letter some fixed number down the alphabet, thus performing a substitution cipher. Ahmas al-Qalqashandi, an egypt writer, introduced the frequency analysis, a method for breaking substitution ciphers, in the 14th century. About 300 years later, the "Geheime Kabinets-Kanzlei" in Vienna routinely intercepts, copies and re-seales diplomatic correspondence to embassies, and manages to decrypt a great percentage of the ciphertexts. In the beginning of the 20th century, the first cryptographic device called "Enigma" [2] is patented for commercial use and is later used in World War 2 by german troops for military communication. Successful attacks against the "Enigma" cipher are demonstrated by polish mathematicians even before outbreak of the war, and systematic encryption of "Enigma" - based ciphertexts are conducted in Bleatchley Park, U.K., by using so called "Turing-Bombs", giving the allies invaluable advantages. The second half of the 20th century brings the introduction of public key cryptography with it: in 1976 Whitfield Diffie and Martin Hellman specify a protocol for key exchange, based on a public key system developed by Ralph Merkle, and one year later, the RSA public key encryption is found by the american mathematicians Rivest, Shamir and Adleman.

Cryptography is basically 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*. Key, clear- and cipher text all are strings built from the alphabet $\mathcal{A}$.

- $\mathcal{A}$ 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

[1] classical greek for *kryptôs*: *concealed*

[2] classical greek for "riddle"

- $\mathcal{C}$ is the ciphertext space, also consisting of the strings from the alphbet $\mathcal{A} = \{0, 1\}$
- $\mathcal{K}$ is called keyspace, also built from the alphabet. Every element $e \in \mathcal{K}$ is called a key and determines the function $\mathcal{M} \rightarrow \mathcal{C}$. This function, E_e is called the *encryption function*.

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

Unauthorized parties - lacking the used key - should, by looking at the ciphertext, learn absolutely nothing about the hidden cleartext beside the length of the origin message. Authorized parties, on the other hand, are able to retrieve the original data out of the ciphertext by using the key with polynomial work, thus reversing the encryption. This reversing process is called *decryption*.

- For every key $d \in \mathcal{K}$, D_d denotes the function from $\mathcal{C} \rightarrow \mathcal{M}$, 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*.

Combining this properties yields a cipher or *encryption scheme*) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, which is a pair of *efficient* [3] algorithms s.t.

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

The correctness property ensures for every pair of $(e, d) \in \mathcal{K}$ and for every message $m \in \mathcal{M}$ that encryption is reverseable, i.e. it must hold that

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

Kerckhoff's Principle

When designing ciphers, a fundamental question is what components of it must be protected from public knowledge, and what parts can be published without compromising the security of the system. A cipher is considered secure if it is not breakable by an adversary in a reasonable time frame [2], where this time frame is a function of the useful timespan of the protected data. This synonymously means that an adversary must spend

[3]"runs in polynomial time"

exponential work. It follows that *every* cipher can be broken in principle by mounting the "brute-focrce" attack, searching the correct n-bit key in the exponential big key space 2^n. Thus, such an exhaustive search must be rendered impracticable by using a suitable large key space to obtain a secure cipher.

The dutch cryptographer Auguste Kerckhoff added some additional rules for designing a secure cipher. According to *Kerckhoff's Principle* stated 1883, among other properties, 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).

This separation of key and algorithm allows the publication of the basic cipher methods, benefiting from peer review. The contradicting approach by trying to hide the inner workings from public to increase security is also known as "Security by Obscurity".

Another definition of a secure cipher was introduced by Shannon in 1949 [3], viewed from a communication-theory point of view. Depending on the message space, there is a finite number of possible cleartext messages, each occurring with its own *a priori* probability. These messages are encrypted and sent to the receiver. An eavesdropper, intercepting messages, can calculate the *a posteriori* probabilities for all possible cleartext messages, leading to the observed cipher text. If booth probabilities are the same, the attacker has learned absolutely nothing from intercepting the cipher text, which is defined by Shannon as *perfect secrecy*. A prerequisite for a perfectly secure cipher is that the key space is at least as big as the message space. Otherwise there will exist cleartext messages which are mapped to the same cipher texts, and thus a priori and a posteriori probabilities will change.

Because all cryptographic schemes rely on the generation of random numbers, a short introduction to probabilistic theory and Pseudo Random Number Generator (PRNG) is given, followed by a introduction to the most important representatives for symmetric and asymmetric ciphers.

2.3 Randomness and Probabilistic Theory

A basic requirement of all cryptographic schemes is the availability of randomness. *Entropy* is the unit of the unpredictability of a process, and was also defined by Shannon. The higher the predictability, or in other words, the more likely an event, the lower its entropy. Flipping a "fair" coin is a canonical example of a process with maximum entropy, because every coin flip hsa a probability of $\frac{1}{2}$, and all flips are independent from each other [4]. If obtaining heads of the coin is viewed as a logical "0" and tails as a logical "1", a binary string of arbitrary length can be built, where the probability of all possible strings of same length is equal, as shown in figure 2.1, yielding a *uniform distribution*.

The importance of random numbers in cryptography is founded on the nature of the cipher used, as will be shown in the next sections. For example, stream ciphers generate a keystream which is used for encryption. If the keystream is predictable by an adversary, the security of the cipher is reduced. Similar arguments are valid for block ciphers, which often rely on an initial value called Initialization Vector (IV) for encryption. As a last example, many protocols rely on determining a random prime number. Again, if such a prime number can be narrowed down within some boarders, this fact may weaken the encryption process.

A fundamental problem in generating random numbers by utilizing computing devices is the deterministic nature of an algorithm:

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." [4]

Such numbers are therefore called *pseudo*random. Lots of cryptographic products suffered serious flaws because of relying on a broken PRNG. A historical example of such a broken random number generator, outputting biased (i.e., not uniformly distributed values) was "RANDU", invented by IBM in the 1960s. The generator belongs to the class of multiplicative congruential algorithms as proposed by Lehmer [5], which can in principle generate random numbers of sufficient quality, *if* the correct parameters are chosen. Random values can be obtained after setting an initial value for I_0, called *seed*, and repeatedly executing the calculation

$$I_{j+1} = 65539 * I_j \pmod{2^{31}}$$

One problem is that consecutive values generated by RANDU are not independent, which can be seen in figure 2.2. To obtain the plot, 10000 uniformly distributed random numbers were chosen as initial seeds for I_i and plotted as x-values. I_{i+1} served as y- and I_{i+2} as z-values. While one would suspect that all points would be equally distributed in space, a clear pattern is visible, indicating that the values are correlated.

To assess the quality of a PRNG, beside of such spectral tests lots of additional tests are available, see [6] for details.

To encounter the shortcomings of a PRNG, a True Random Number Generator (TRNG) uses a natural process as non-deterministic data source, for example thermal noise of a semi conductor, cosmic noise from space or digital oscillators.

[4] John von Neumann, 1951

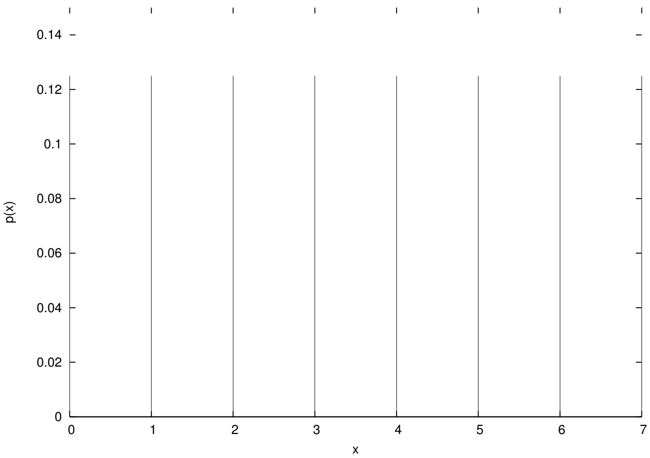


Figure 2.1: Uniform Distribution of binary string of length 3

The used hardware platform, the RapsberryPi, offers a hardware number generator - the quality of its provided random numbers will be subject to various statistical tests, see chapter **??** for the results.

2.4 Symmetric vs. Asymmetric Cryptography

As stated above, two very fundamental differences regarding the key used in a cryptographic system can be found. Symmetric ciphers, where in the most cases the same key is used for encryption and decryption, outperform its asymmetric counterparts in regards of data throughput by a factor of about 1000 [7]. Additionally, they need shorter keys to achieve the same level of security - both arguments encourage it's use in embedded devices because of its less computing and memory demands.

The big disadvantage of symmetric ciphers is that the key must be known to sender and receiver of the message *before* secure communication can take place. This constitutes some kind of chicken-egg problem: to be able to send encrypted data, the key must be distributed, i.e. a secure channel has to be setup first, but the key can obviously not be sent before this secure channel exists.

Asymmetric or public key cryptography solves the problem of key distribution by using two different keys, belonging to the same key pair: the *private* key must be be protected from disclosure, while the *public* key can be published without harming security. For encryption, the public key of the receiver is used, who in turn will use his private key to decrypt the message.

To be able to take benefit from the advantages of booth schemes, a hybrid approach

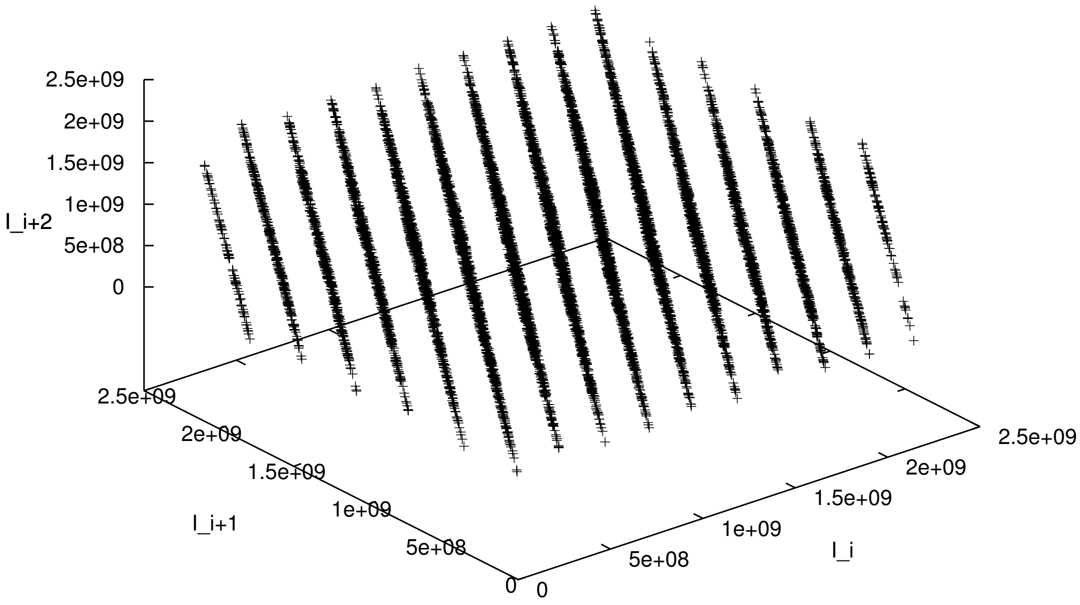


Figure 2.2: Spectral Plot of RANDU output

is possible: at first, public key cryptography is used to negotiate a symmetric session key, which then can be used to encrypt the actual, sensitive data.

Stream Ciphers

Stream ciphers belong to the family of symmetric ciphers, thus $e_i = d_i$. 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: $m_i = D_d(d_i, c_i)$.

The Vernam Cipher

This cipher, also called One Time Pad (OTP), was invented by Gilbert Vernam in 1918, and belongs to the family of polyalphabetic stream ciphers, which means that every character of the origin message is mapped to another character of the same alphabet. In contrast to a monoalphabetical cipher, there is no fixed mapping between the input and output characters. The substitution is achieved by generating a keystream and by executing a bit-wise XOR operation, as defined in table 2.4, of key and message.

		$\bigoplus$
0	0	0
0	1	1
1	0	1
1	1	0

Obviously, the security of the cipher heavily depends on the quality of the PRNG. If a truly random source is used to generate the key stream, this cipher has perfect secrecy. Nevertheless, the cipher can be completely broken if the same key is used for encrypting more than one cleartext message, allowing to mount an attack based on frequency analysis. Imagine an attacker is able to intercept a specific amount of different ciphertexts, all encrypted with the same key. Pairwise xor'ing of the ciphertexts yields the xor-combination of the corresponding cleartexts, because

$$m_1 \bigoplus m_2 = (c_1 \bigoplus k) \bigoplus (c_2 \bigoplus k) = c_1 \bigoplus c_2 \bigoplus k \bigoplus k = c_1 \bigoplus c_2 \bigoplus 0 = c_1 \bigoplus c_2$$

Whenever the same character is present in two different ciphertexts at the same position, the result of the xor operation will be 0x00, allowing to draw inferences about the language used. By utilizing frequency analysis, the used key can be determined position by position with effort bounded by $O(n^2)$.

Stream Ciphers based on Linear Feedback Shift Register (LFSR)

An disadvantage of the Vernam cipher is that a key of equal length as the message is necessary. To mitigate this problem, a LFSR can be used to generate a key of proper length from a much short, initial key. Figure 2.3 shows a 4 stage LFSR

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[5]. A disadvantage of block ciphers is that it

[5] Counter Mode, see 2.4

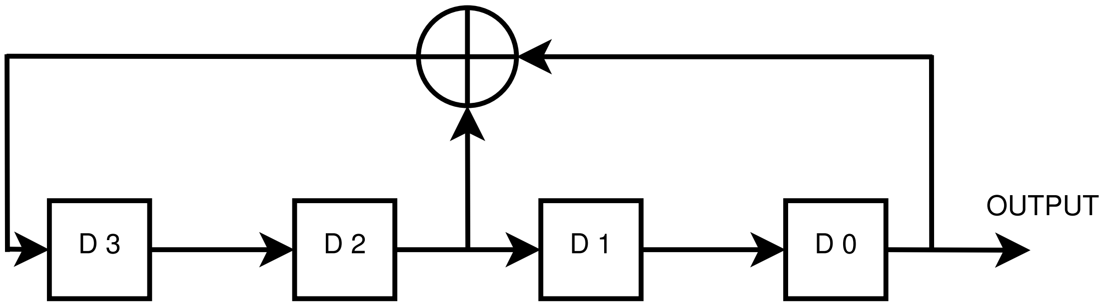


Figure 2.3: 4 Stage LFSR

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

Permutation Blockciphers

Substitution Blockciphers

Product Blockciphers

Feistel Networks

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.

$$C_0 = E(k, (M_0 \bigoplus IV))$$
$$C_1 = E(k, (M_1 \bigoplus C_0))$$
$$\dots$$

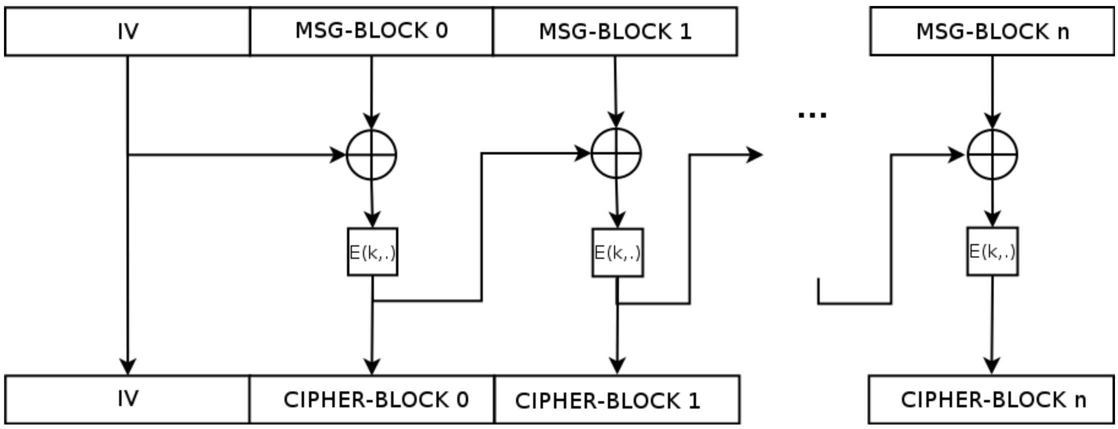


Figure 2.4: Cipher Block Chaining for encrypting messages

$$C_i = E(k, (M_i \bigoplus C_{i-1}))$$

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

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

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 2.5. 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).

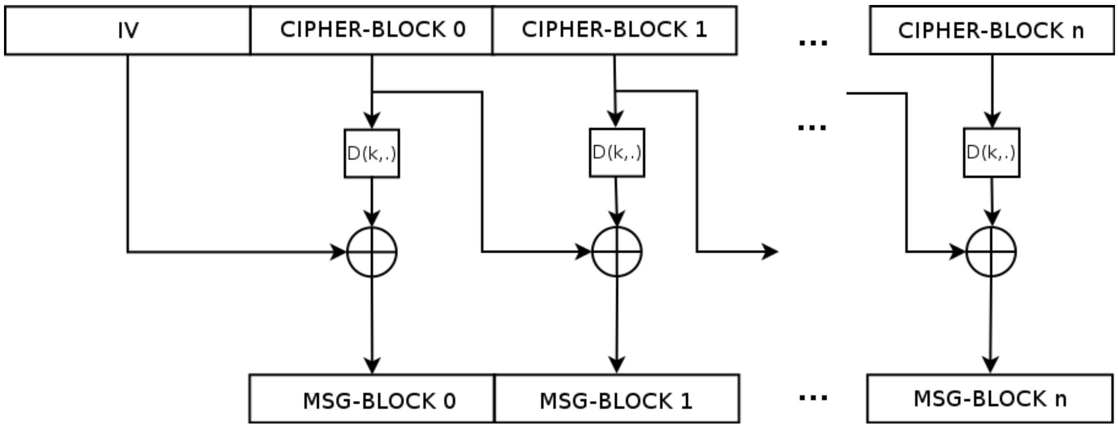


Figure 2.5: Cipher Block Chaining for decrypting messages

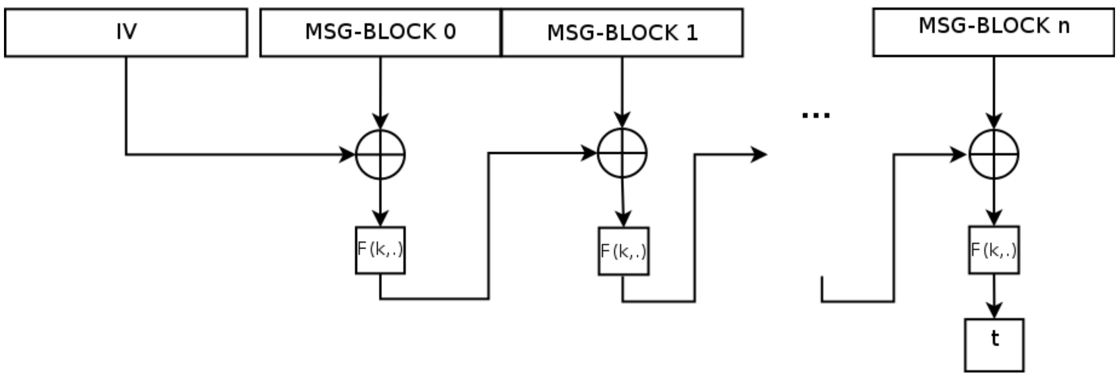


Figure 2.6: Cipher Block Chaining for generating a MAC

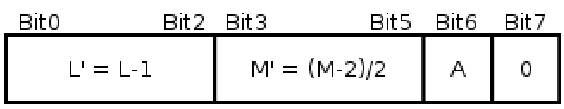


Figure 2.7: Flag Field of CBC IV

Counter Mode - CTR

Authenticity

OCB

Cipher Block Chaining - CBC

2.5 Authenticated Encryption

CCM

CCM[6], 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 2.6.

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 2.7. 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 2.4 in figure2.6, the first message block M_0 is xor'd with a nonce or initialization vector(IV, see figure 2.8), 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

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

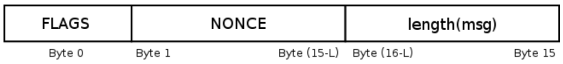


Figure 2.8: IV for CBC MAC

becomes the input for the block cipher, and so on, iterating over all n message blocks to determine the tag t:

$$C_0 = F(k, M_0 \bigoplus IV)$$
$$C_1 = F(k, M_1 \bigoplus C_1)$$
$$\ldots$$
$$C_n = F(k, M_n \bigoplus C_{(n-1)})$$

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 2.9.

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

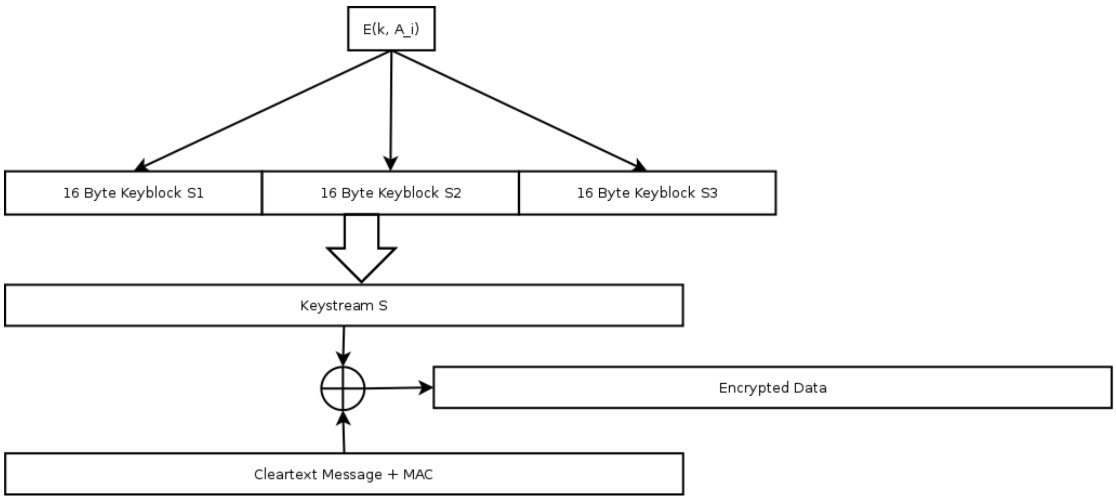


Figure 2.9: CTR Encryption

2.6 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

2.7 Attacks on Ciphers

Passive Attacks

timing attacks - constant time computation

Active Attacks

BLABLA:

Such a cipher as defined above provides confidentiality, i.e. it ensures that only authorized parties are able to decrypt the message. This leads to other problems, namely how to determine who is authorized, i.e. how to provide authenticity, and how to assure that the message was not altered when, i.e. how to provide integrity. It turns out that such a cipher is suitable for these purposes

A system is an entity that interacts with other entities, which constitute the environment for the system and can be other systems, humans or the physical world [8]. Fundamental properties of communication systems are *functionality, performance, security and dependability*. The system provides services to the user(s) of the system through it's service interface, described by the functional specification. Whenever the provided service deviates from correct service a system failure occurs. An informal definition of a dependable system is a system which delivers a service that can be justifiable trusted. More formally, dependability consists of the following attributes: *Availability*, which means that the system is ready for correct service, *reliability*, the continuity of correct service, *safety*, i.e. the avoidance of catastrophic consequences *integrity*, s.t. the system cannot be modified in an unwanted manner and *maintainability*, so that the system can be repaired in the case of a failure.

In case of a secure system, another important property is *confidentiality*, which means that no information is disclosed to unauthorized entities. To achieve

Finite fields

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).

2.8 Propabilistic Theory

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

2.9 Security in HBA

CHAPTER 3

KNX

3.1 Introduction

Konnex (KNX) implements a specialized form of automated process control, dedicated to the needs of HBA. KNX [1] emerged from 3 leading standards namely the European Installation Bus (EIB), the European Home Systems Protocol (EHS) and BatiBUS. It is an open, platform independent standard, developed by the KNX association implementing the EN50090 standard for home and building electronic systems.

To provide platform independence, the standard uses a layered structure, based on the International Organization for Standardization (ISO) / Open System Intercommunication Model (OSI). Different kind of physical backends are supported, allowing it to be used in different scenarios.

EIB already supported interoperability between products from different manufacturers. This was achieved by the definition of the EIB interworking standard (EIS), which standardizes the data transported inside the datragrams. KNX continued this efforts with the introduction of common data types (DT), distinguishable through unique ids, thus standardizing their encoding, format, range and unit. Every DT groups related data point (DPT), the actual control variables of the network, together, allowing

For example, every KNX certified manufacturer producing a switching actuator must use the defined dataformat - an end-user can therefor exchange such an actuator without caring about compatibility issues. For configuration and parametrization of the devices, a Windows based software suite called engineering tool software (ETS) is used, which also offers a bus monitor for debugging.

[1] connexio, latin for connetion

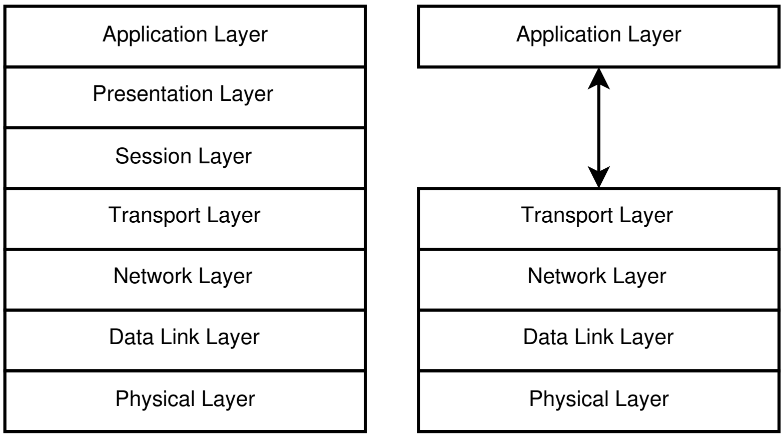


Figure 3.1: OSI Layer Model, compared to the KNX Modell

3.2 KNX Layers

The OSI standardizes the communication between different, independent systems by grouping the needed functions into 7 sublayers to provide interchangeability and abstraction. Every layer provides services to its next-higher layer, and uses the services provided by its next-lower layer. Every service is defined by standardized interfaces - that way any layer can be modified internally without compromising the function of the system, as long as the defined interfaces are implemented. This fragmentation of one service follows the paradigm of *impera et divide*[2] and facilitates the building of complex systems by dividing one complex problem into subsequent, less-complex problems.

KNX implements this model, omitting layers 5 and 6, as shown in figure 3.1. Data from applications are directly passed to the transport layer in a transparent way, and vice versa.

Physical Layer

This is the lowest layer as defined by OSI and determines the basic transmission parameters like symbol rate, signal form but also mechanical characteristics like which connectors are used.

To provide flexibility in KNX, 4 different physical media are defined. Twisted Pair (TP)-1 which was inherited from EIB, and is the successor of TP-0, as defined by BatiBUS, 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. TP-1 uses collision avoidance, and allows all topologies beside rings.

[2] latin: *dive and rule*

Because this work is base on the TP1 - part of KNX only, this medium will be explained in more detail in the next section.

PL110, which was also inherited from EIB, uses power line installations for communications. The carrier uses spread frequency shift keying, and can be used for bidirectional, half duplex communication with an even lower data rate of 1200 bit/sec. KNX Radio Frequency (RF) is used for short range wireless communication at 868,3 MHz. KNXnet/Internet Protocol (IP) allows the integration of KNX into networks using Transmission Control Protocol (TCP) / IP for communication. Here, 3 different communication modes are defined: *tunneling* mode is used for configuration and monitoring a client device by a KNXnet/IP server. Routing mode is used for connecting KNX lines over IP, while KNX IP is used for direct communication between KNX devices. [9]

TP-1

The accurate name for this medium is 'Physical Layer type Twisted Pair', with variants PhL TP-1-64 and PhL TP-1-256, which is backward compatible to the former one. While the first one allows the connection of up to 64 devices, the latter one allows up to 256 devices connected in a linear, star, tree or mixed topology as one physical segment, also called a *line*.

Bridges do not possess their own address and are used for galvanic separation of physical segments and for extension of TP-1-64 segments to allow up to 256 devices. Therefore, they acknowledge layer 2 frames received on one side and forward them to their second interface.

Routers have their own address space and only forward packages received on one side if the destination address is located on the other side of the router. As well as bridges, they can be used for galvanic separation and they acknowledge frames on layer 2. A line coupler (LC) is a router that integrates up to 16 lines into one logical object called *area*. A backbone coupler (BbC) is a router that connects up to 16 areas to one network, thus providing the maximum size of a network consisting of 65536 devices:

- up to 256 devices per line
- up to 16 lines per area = 4096 devices in 16 lines
- up to 16 areas for whole network = 65536[3] devices in 16 areas

Gateways are used to connect KNX networks to non-KNX networks.

A logical '1' is regarded as the idle state of the bus, so the transmitter of the Medium Access Unit (MAU) is disabled when sending a '1', i.e. the analog signal on the bus

[3]it is to be noted that the actual number of usable devices is smaller because routers have their own address

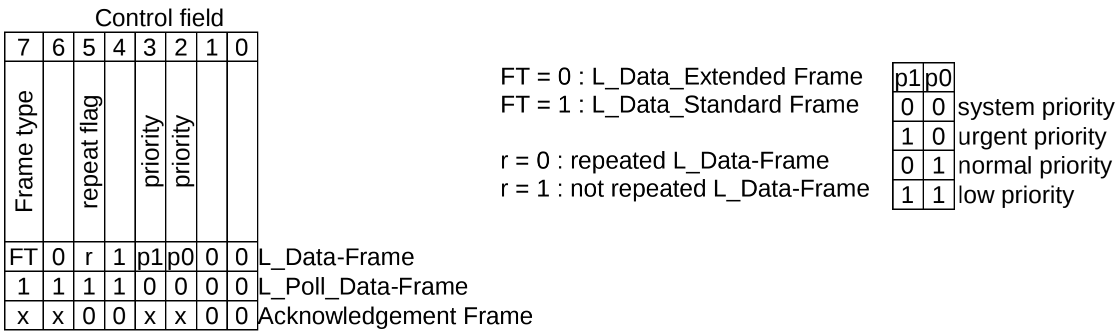


Figure 3.2: Control Field

consists only of the DC part. TP-1 uses courier sense multiple access (CSMA)/collision avoidance (CA) for bus access, so every devices must listen to the bus and is only allowed to begin sending when the bus is idle. In the case of a simultaneous transmission start, a logical '1' of one device 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. This behavior is be used to implement priority control and is exploited by the next layer.

Data Link Layer for TP1

This layer is responsible for error detection, retransmission of corrupted packages, framing of the higher level packages into suitable frames and accessing the bus according to the rules used by the particular bus medium. It is often broken into 2 distinct sublayers, namely the Medium Access (MAC) as bus arbiter and the logical link control (LLC), providing a reliable point-to-point datagram service.

Three frame formats are defined: L_Data frames are used for sending a data payload to an individual address, a group address or for broadcasting data to the bus. L_Poll_Data frames are used to request data from an individual knx device or a group of devices. Acknowledgement frames are used to provide a reliable transport mechanism, i.e. to acknowledge the reception of a frame by a knx device.

For L_Data_Frame, 2 different formats are defined: standard frames, as shown in figure 3.3 and extended frames, see figure 3.5. While standard frames can bear up to 15 bytes of application data, extended frames allow the transmission of up to 254 bytes of data.

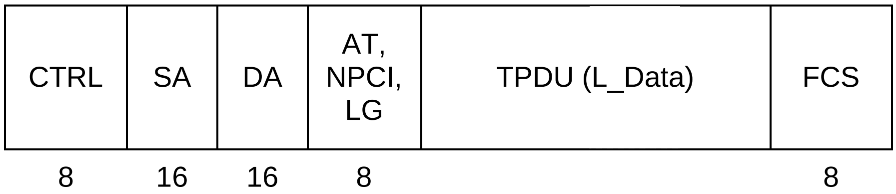


Figure 3.3: Standard Frame

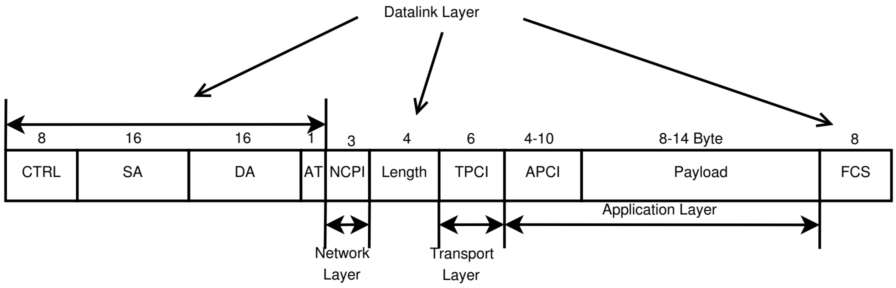


Figure 3.4: Standard Frame, in detail

Standard L_Data_Frame

Every standard frame starts with the control field, determining the frame type. After that, sender address and destination address, each 2 byte, follow. The next byte contains 1 address type bit, 3 bits which belong to the Link Service Data Unit (LSDU) of the next higher layer and 4 bits of length information, resulting in an maximum payload of 15 bytes(by design, it is also allowed to set this length field to 0, i.e. to send an empty data frame). After the corresponding number of payload bytes, a check byte completes the frame. This check frame is defined as an odd parity over all preceding bytes, which represents a logical NOT XOR function.

Extended L_Data_Frame

The extended frame starts with a control field, as a standard frame. After that, a special Extended Control Field (CTRLE) follows, as shown in figure 3.6. Source- and destination addresses, each 2 bytes, follow. To allow the bigger payload, the next byte is used as length field, with the value 0xFF reserved as escape code. After that, the payload and the check byte, as defined above, follow.

L_Poll_Data Frame

These frames serve as data requests of the poll-data master for a maximum of 15 bytes and start with a control field, as defined, followed by the 2 byte source address of the

CTRL	CTRLE	SA	DA	LG	TPDU (L_Data)	FCS
8	8	16	16	8		8

Figure 3.5: Extended Frame

Extended Control field

7	6	5	4	3	2	1	0	
Address Type	Hop Count			Extended Frame format (EFF)				
AT	r	r	r	t	t	t	t	
0	r	r	r	0	0	0	0	Point-to-point Addressed L_Data_Extended Frame
1	r	r	r	0	0	0	0	Standard Group addressed L_Data_Extended Frame
1	r	r	r	0	1	x	x	LTE-HEE extended address type
								All other codes are reserved for future use

Figure 3.6: Extended Frame CTRLE field

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 by transmitting the databytes in the corresponding poll slave slot. This is achieved by defining exact timings when each poll data slave must send the requested data. Therefor, such frames can only be used within one physical segment [10].

Acknowledge Frame

This frames are used to acknowledge the reception of a knx data frames(FIXME: ONLY DATAFRAMES??) for group- or individual addresses and consist of one byte, sent after a fixed timespan after reception of the frame.

KNX addressing scheme

Two different kinds of addresses are defined: group- and individual addresses, which type is used is determined by the 'address type' flag in the control field of the datagram(0 = individual address). While the source address always is an individual address and must be unique within the network, the destination address can be of type group or individual (see figure 3.7 for the layout).

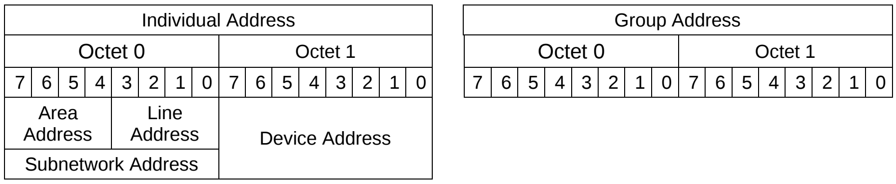


Figure 3.7: KNX individual and group addresses

For poll-data frames, the destination address determines the *poll group address* which must be unique within the physical segment.

Poll data responses as well as acknowledgement frames each just contain 1 byte, i.e. they do not possess source- and destination addresses.

Network Layer

The main task of the network layer is the routing and forwarding of packets, so the main parameter on this layer is the destination address of the datagram. Additionally, KNX reserves 3 bits of every standard- and extended data frame as *hop count*. This counter is decremented by every router and the frame is discarded if the counter reaches the value zero. This mechanism, known from Internet Protocol version 4 (IPv4) [11] [4], avoids the infinite circulation of packages within an incorrectly configured network.

Transport Layer

Accoriding to the OSI modell, this layer provides point-to-point communication for hosts.

In KNX, the connection orientated, reliable point-to-point communication mode addresses the individual address of a remote device and uses a timer to detect timeouts. Up to to 3 retransmissions are allowed if the sent datagram is not acknowledged. A simple handshake - similar to transport control protocol (TCP) - is used, as shown in figure. 3.8.

All other modes are unreliable, i.e. unacknowledged, transport mechanisms and can be used to address individual addresses, group addresses or all devices in the network. For the latter mode, the special KNX broadcast address 0x0000 is reserved. The transport layer protocol control information (TPCI), included in the control field, determines the type of the transport layer protocol data unit (TPDU) and also posseses a 4 bit sequence number by which duplicate datagrams, caused by damaged acknowledge-frames, can be discarded.

[4] Originally, this was called time-to-live (TTL)

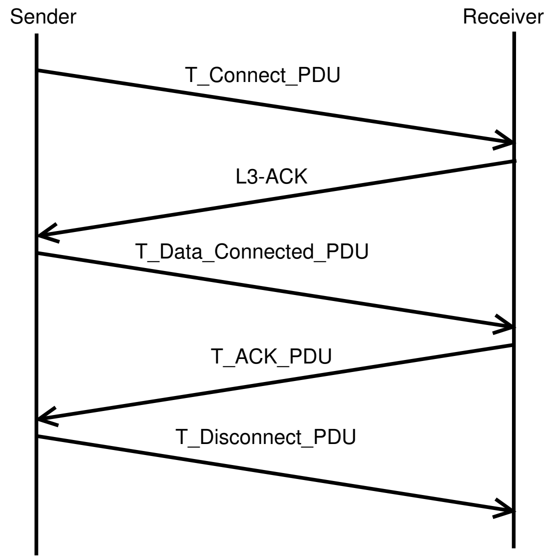


Figure 3.8: Handshake for connection-orientated communication

Octet 5								Octet 6								
								transport ctrl field								
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
Address Type (AT)								Data/Control Flag	Numbered							
1								0	0	0	0	0	0			T_Data_Broadcast-PDU (destination_address = 0)
1								0	0	0	0	0	0			T_Data_Group-PDU (destination_address <> 0)
1								0	0	0	0	0	1			T_Data_Tag_Group-PDU
0								0	0	0	0	0	0			T_Data_Individual-PDU
0								0	1	SeqNo	SeqNo	SeqNo	SeqNo			T_Data_Connected-PDU
0								1	0	0	0	0	0	0	0	T_Connect-PDU
0								1	0	0	0	0	0	0	1	T_Disconnect-PDU
0								1	1	SeqNo	SeqNo	SeqNo	SeqNo	1	0	T_ACK-PDU
0								1	1	SeqNo	SeqNo	SeqNo	SeqNo	1	1	T_NAK-PDU

Figure 3.9: Flags used at the Transport Layer

Session Layer

This layer is responsible for maintaining sessions, i.e. it provides services to maintain synchronized data exchange. It does not exist in KNX.

Presentation Layer

This layer allows a system-independent data representation, which is not necessary in KNX because the use of standardized DT.

Application Layer

This layer provides services for process-to-process information through a KNX network. Up to 10 bits are reserved in the application control field, inside the application layer protocol data unit (APDU), containing the application layer service code. The provided services range from tasks like reading or writing group values, distribution of network parameters to obtaining device information.

3.3 KNX security concept

In the early days of HBA, communication security was not considered a critical requirement: firstly, the communication was done over wires, i.e. physical access to the network would have been necessary for attacking the network. Secondly, the possible threats by misusing applications like lights- or shutters-switching were considered negligible. Additionally, the devices used in such networks were characterized by very limited processing power - thus, the comprehensive use of encryption would have put remarkable computing loads onto these devices and was therefore considered impracticable.

The basic KNX standard therefor does not specify any security mechanisms for control information:

"For KNX, security is a minor concern, as any breach of security requires local access to the network. In the case of KNX TP1 or KNX PL110 networks this requires even physical access to the network wires, which in nearly all cases is impossible as the wires are inside a building or buried underground. Hence, security aspects are less of a concern for KNX field level media." [12]

For KNX/IP, the physical containment arguments do not apply. To counter this, it is proposed to use firewalls and Virtual Private Network (VPN) to prevent unauthorized access, as well as hiding critical network parameters from public. The latter concept is also known as *"security by obscurity"*, offering - if at all - only little protection.

For management communication, a rudimentary, password-based control is used. Therefore, KNX suffers the following security flaws [13]: for management, the used keys are transmitted as cleartext, enabling an attacker to perform a passive attack to obtain the password. Subsequently, the attacker can mount an active attack, injecting arbitrary management messages. No methods are foreseen for generation or distribution of the keys. For control information, an adversary can directly inject arbitrary messages to control the network, allowing passive and active attacks too. These shortcomings clearly disqualified KNX for usage in critical environments, restricting its possible field of application.

Today, HBA systems are used on a large scale, and the available processing power on embedded computing platforms has risen significantly, so the deployment of such systems would be possible also in critical environments, under the condition that proper security mechanisms are deployed. For KNX, several extensions exist which will be introduced in the next sections.

KNX Data Security

In 2013, the KNX association published "Application Note 158" [14] which specifies the KNX Secure-Application Layer (S-AL), providing authentication and encryption, and the Application Interface Layer (AIL), implementing access control, booth being part of the application layer. The settlement of these functions above the transport layer allows a transparent, communication media independent end-to-end encryption.

The application layer service code 0xF31 is reserved for this purpose, indicating that a secure header and a S-AL protocol data unit (PDU) follow instead of a plaintext-PDU. This allows the flexible usage of the secure services just in situations where they are needed - otherwise, the plaintext application layer services can be used.

The S-AL services defines modes for authenticated encryption or authentication-only of a higher-level cleartext APDU. As underlying block cipher Advanced Encryption Standard (AES)128 is used in Counter with CBC MAC (CCM) mode, encrypting the payload with Counter Mode (CTR) and providing integrity with Cipher Block Chaining (CBC) mode. The overhead introduced by the Message Authentication Code (MAC) is reduced by using only the 32 most significant bits instead of the whole 128 bit block obtained from CBC. Source- and destination address as well as frame- and addresstype, the TPCI, length information and a 6 byte sequence number determine the IV for the CBC algorithm and are therefor also protected by the MAC. The sequence number is a simple counter value that provides data freshness, thus preventing replay attacks, and is sent along with every S-AL PDU. For synchronization of this sequence number between two devices, a S-AL s ync-service is defined. Because no sequence number can be used here to guarantee data freshness, a challenge-response mechanism is used instead. Two different types of keys are used: a Factory Default Setup Key (FDSK) is used for initial setup with the ETS. The ETS then generates the Tool Key

(TK), which is used by the device for securing of the outgoing messages. Consequently, every device must know the TK of it's communication partners.

While the S-AL empowers two devices to communicate in a secure way, the AIL allows a fine-grained control which sender has access to which data objects. Therefore, every *link* (a combination of source address and data or service object) is connected with a *role*, which in turn has some specific *permissions*.

EIBsec

EIBsec is another extension to KNX, providing data integrity, confidentiality and freshness, allowing its deployment in security-critical environments. A semi-centralized approach is taken here by using special key servers, responsible for dedicated sets of keys, providing a sophisticated key management. EIBsec divides a KNX network into subnets, connected by devices called Advanced Coupler Unit (ACU). Beside their native task, i.e. routing traffic, they are responsible for the key management of their network segments, which includes key generation, distribution and revocation. Every standard device that wishes to communicate with other devices must at first retrieve the corresponding secret key from its responsible ACU, which can therefor control the group membership of the requesting device by allowing or denying the request respectively by revoking the key at a later point in time.

EIBsec uses two different keys: in normal mode, a session with the keyserver is established to retrieve the session key, establishing a secure channel. This mode uses encryption-only by utilizing a Pre Shared Key (PSK), integrity must therefor be guaranteed by the sender of the message. Counter mode is used for transmitting management and group data over the secure channel. A simple Cyclic Redundancy Check (CRC) is added to the payload before encryption and shall guarantee integrity. Booth modes encrypt the traffic on transport level, allowing standard routers to handle the datagrams. As block cipher, AES-128 is used in CTR mode.

FIXME: zitieren / originalpaper...?

KNX IP Security

This work focuses on securing the KNX IP specification, which can be used as backbone for connecting distinct KNX installations [15]. Thanks to the widespread use of TCP/IP, a wide range of physical transport mechanisms can be utilized.

A structure comparable to the design of Transport Layer Security (TLS) is defined by building a distinct security layer, residing above the transport layer (therefore, it directly connects the transport to the application layer, because session- and presentation layer are empty, as defined by the KNX specifications, see chapter 3).

The design distinguishes 3 different types of modes:

in the *configuration phase*, every device that wants to participate in the secure network generates an asymmetric key pair, which is sent to the ETS over a secure channel (for example, by transmitting the data over a direct, serial connection between the ETS host and the KNX device). The ETS, acting as Certification Authority (CA), signs the combination of IP and public key with its own private key, thus generating a certificate, which is sent back to the device, along with the public key of the ETS.

After that, the *key set distribution phase* starts, where a unicast and a multicast scenario are differentiated: in the unicast case, the device initiating the communication - called client - obtains the key set from the target device by a 2-step handshake: at first, mutual entity authenticity is established by utilizing the certificate provided by the ETS. Afterwards the keyset is obtained from the target, which concludes the second phase.

In the last step, secure communication can take place, i.e. the the client is able to encrypt the data with the obtained key and sends it to the target device.

For the multicast scenario, a distinct *coordinator*, responsible for maintaining the group key, is necessary. Every powered-up device identifies the coordinator as soon as possible by broadcasting *hello* requests, adopting the coordinator role if no replies are received in time. To actually send a payload, the group key is obtained from the coordinator and the data is sent to the group, analog to the unicast case. By adding mechanisms to detect "dead" coordinators and delegating the coordinator role to a different device, the design avoids a Single Point of Failure (SPOF).

3.4 Summary

CHAPTER 4

Solution

4.1 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 4.1.

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

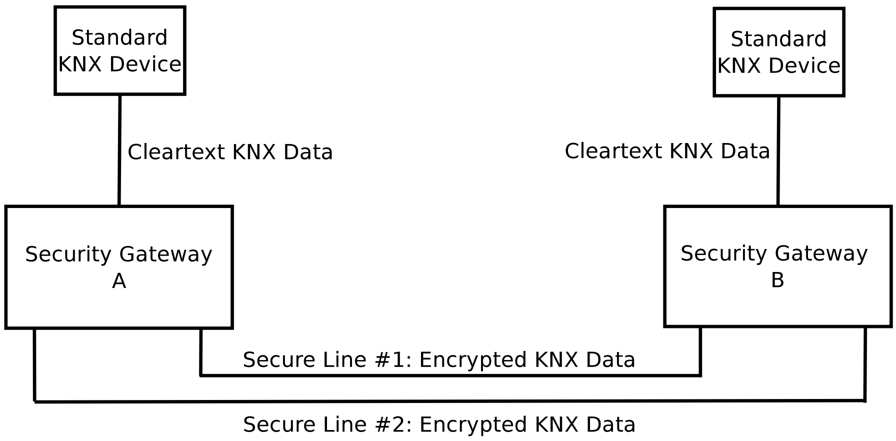


Figure 4.1: Secure Area

As stated in chapter 3, 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 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 4.3 and 4.4, 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 4.2 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

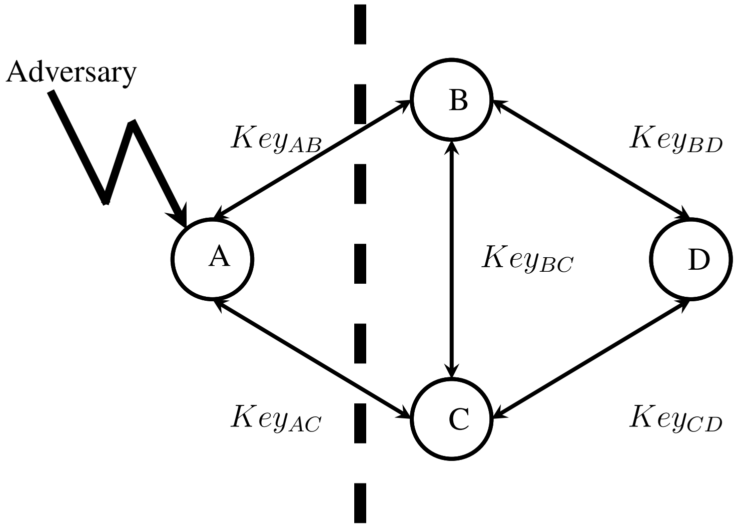


Figure 4.2: Firewall

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 client device in the first place. One message must be discarded to avoid duplicates. This is achieved with a monotonically increasing counter that also guards against replay attacks: whenever a package, generated by a client, enters the network, a counter for outgoing packages is incremented which must be sent along the duplicated packages so that the receiving side can discard one of the 2 packages. This counter must be unambiguous for every source/destination address tuple of the origin cleartext message. The receiving side must maintain a counter for incoming packets, also parametrized by source and destination address. If booth lines are available, one message will be handled first and trigger an incrementation of the corresponding source/destination counter. The duplicated message, which is handled after that, can safely be discarded because the corresponding counter value will be less than the saved value. Nevertheless which package from which secure line is forwarded to the unsecured line, each line must acknowledge every received package: this is done by generating a special acknowledge frame which is sent back to the sending gateway. The payload of this package must allow the sending gateway to unambiguously identify the acknowledged package, i.e. it must bear source and destination address of the package generated by the client, as well as the used counter value. As a consequence, these acknowledgement frames must be encrypted and authenticated as well. If no acknowledgement frame is received within time t_{ACK}, a retransmission is done on booth lines. This retransmission simply resubmits the same package with the same counter value again. Regarding the security this is no problem because a passive attacker can not learn anything from such a repeated package.

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 3, 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 3.

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

4.2 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

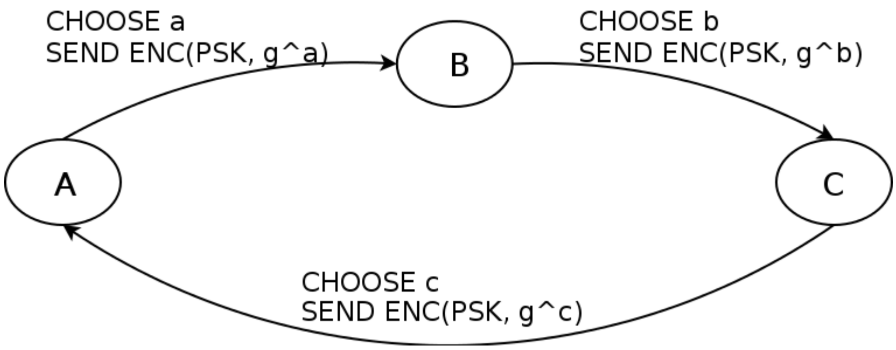


Figure 4.3: DH Round 1

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** 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 4.3:

   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 4.4

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

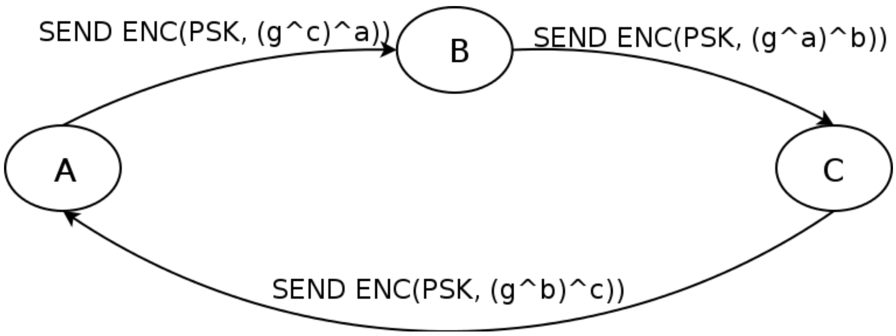


Figure 4.4: DH Round 2

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 4.5:
   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 $s = (g^a)^b = g^{a*b}$

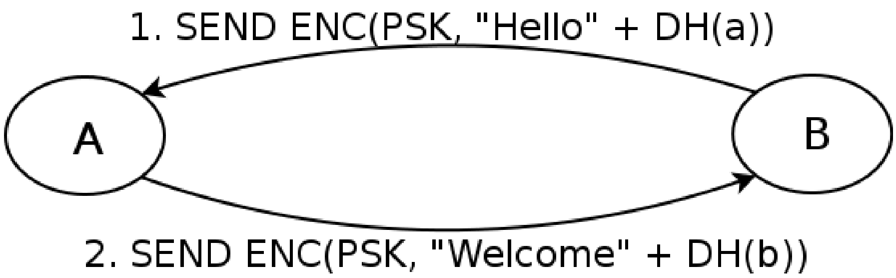


Figure 4.5: DH 2 Parties

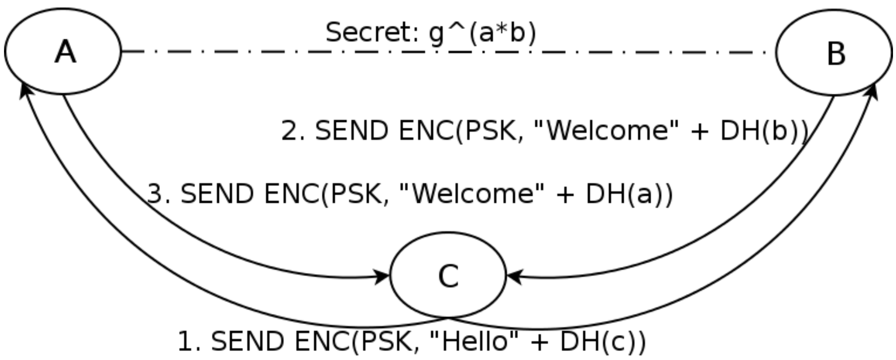


Figure 4.6: DH 3 Parties

3. if new device C want to join, see 4.6
   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.

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 5

Implementation

5.1 Master daemon

KNX addressing scheme

Care must be taken that no duplicate knx addresses are used within the network. Therefore, the following addressing convention is proposed: While it would be possible to use the same addresses on booth lines per gateway, a different scheme is used. For the secured network, the address ranges starting at address 1.1.1 to address 1.1.15 and 1.2.1 to 1.2.15 are reserved for secure line number 1 and 2 respectively, which allows a maximum of 15 gateways. Different addresses are used mainly because it facilitates debugging. Additionally, the used address ranges can be re-used outside the secured network by standard devices anyway. On the unsecured lines, every gateway uses an address from the range 1.0.1 - 1.0.15. Addresses are assigned in a linearly ascending way, so gateway number 1 uses addresses 1.1.1 and 1.2.1 for secure lines 1 and 2, and 1.0.1 for its unsecured line.

APPENDIX A

Setup of the base system

The base system consists of raspbian pi board running the raspbian operating system(a Debian variant), 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.

A.1 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:

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

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

```
[1004111.533698] sdb: detected capacity change from 7909408768 to 0
[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**:

```
# 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 B.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>
```

A.2 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.

```
# tar -xvzf pthsem-2.0.8.tar.gz
# cd pthsem-2.0.8
# ./configure
# make
# 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:

```
# tar -xvzf bcusdk-0.0.5.tar.gz
# cd bcdusk-0.0.5
# ./configure --without-pth-test --enable-onlyeibd --enable-tpuarts
# make
# 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.

A.3 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 A.3 fetches the latest version and creates a directory called 'knxSec' which contains all the needed source files, a proper makefile B.3 for the project, as well as all other needed files.

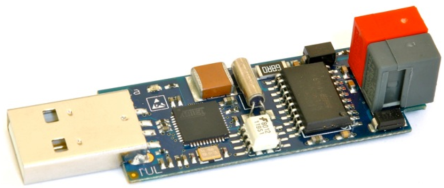
```
# git clone git@github.com:hglanzer/knxSec.git
```

A.4 Busware 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 A.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 A.1 and look for output similar to listing A.4 when plugging the coupler into an USB slot.

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

Figure A.1: Busware KNX-USB coupler



If no such line like 7 appears, the correct firmware is available as file *firmware/TPUARTtransparent.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 A.4 must be executed.

```
# apt-get install dfu-programmer
# dfu-programmer atmega32u4 erase
# dfu-programmer atmega32u4 flash TPUARTtransparent.hex
# dfu-programmer atmega32u4 reset
```

A.5 UDEV

To obtain a consistent naming scheme for the busware dongles, udev rules are provided. This way it is possible to always use the same device file for the distinct bus lines, no matter in which ordering the dongles are connected to the raspberry.

A.6 Test setup

The test environment consists of 2 raspberry pis, as shown in figure 4.1.

APPENDIX B

Code snippets and configuration files

Listing B.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 B.2: Raspbian configuration for dynamic ip address

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

Listing B.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

clean:
	clear
	rm -rf *.o
	rm -f master

run: all
	./master

update:
	git commit -a --allow-empty
	git pull
	git push
```

Glossary

ACU Advanced Coupler Unit. 30

AES Advanced Encryption Standard. 29, 30

AIL Application Interface Layer. 29, 30

APDU application layer protocol data unit. 28, 29

API Application Programming Interface. 49

BbC backbone coupler. 22

CA Certification Authority. 31

CA collision avoidance. 23

CBC Cipher Block Chaining. 29

CCM Counter with CBC MAC. 29

CRC Cyclic Redundancy Check. 30

CSMA courier sense multiple access. 23

CTR Counter Mode. 29, 30

CTRLE Extended Control Field. 24

DPT data point. 20

DT data types. 20, 28

EHS European Home Systems Protocol. 20

EIB European Installation Bus. 20–22

EIBD European Installation Bus Daemon. 49

EIS EIB interworking standard. 20

ETS engineering tool software. 20, 29, 31

FDSK Factory Default Setup Key. 29

HBA home and building automation. v, 19, 20, 28, 29

IP Internet Protocol. 22, 28, 30, 31

IPv4 Internet Protocol version 4. 26

ISO International Organization for Standardization. 20

IV Initialization Vector. 8, 29

KNX Konnex. 20–22, 26, 28–31

LC line coupler. 22

LFSR Linear Feedback Shift Register. 11

LLC logical link control. 23

LSDU Link Service Data Unit. 24

MAC Message Authentication Code. 29

MAC Medium Access. 23

MAU Medium Access Unit. 22

OSI Open System Intercommunication Model. 20, 21, 26

OTP One Time Pad. 11

PDU protocol data unit. 29

PRNG Pseudo Random Number Generator. 7, 8, 11

PSK Pre Shared Key. 30

RF Radio Frequency. 22

S-AL Secure-Application Layer. 29, 30

SPOF Single Point of Failure. 31

SSH Secure Shell. 47

TCP Transmission Control Protocol. 22

TCP transport control protocol. 26, 30

TK Tool Key. 29, 30

TLS Transport Layer Security. 30

TP Twisted Pair. 21–23

TPCI transport layer protocol control information. 26, 29

TPDU transport layer protocol data unit. 26

TRNG True Random Number Generator. 8

TTL time-to-live. 26

URL Uniform Resource Locator. 49

USB Universal Serial Bus. vi, 49, 50

VPN Virtual Private Network. 28

Bibliography

[1] W. Stallings. "Computer Security". In: New Jersey: Pearson Education International, 2008. Chap. Overview.

[2] J. Menezes, P. van Oorschot, and Vanstone S. "Handbook of Applied Cryptography". In: New York: CRC Press, 1997. Chap. Overview of Cryptography.

[3] C.E. Shannon. "Communication theory of secrecy systems". In: *Bell System Technical Journal, The* 28.4 (1949), pp. 656–715. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

[4] R. Gennaro. "Randomness in cryptography". In: *Security Privacy, IEEE* 4.2 (2006), pp. 64–67.

[5] D. H. Lehmer. "Mathematical methods in large-scale computing units". In: *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery, 1949*. Harvard University Press, Cambridge, Mass., 1951, pp. 141–146.

[6] National Institute of Standards and Technology. *A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications @ONLINE*. 2010. URL: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf.

[7] M.A. Matin et al. "Performance evaluation of symmetric encryption algorithm in MANET and WLAN". In: *Technical Postgraduates (TECHPOS), 2009 International Conference for*. 2009, pp. 1–4.

[8] A. Avizienis et al. "Basic concepts and taxonomy of dependable and secure computing". In: *Dependable and Secure Computing, IEEE Transactions on* 1.1 (2004), pp. 11–33. ISSN: 1545-5971.

[9] Lukas Krammer et al. "Security Erweiterung fuer den KNX Standard". In: *Tagungsband – innosecure 2013 – Kongress mit Ausstellung fuer Innovationen in den Sicherheitstechnologien Velbert Heiligenhaus*. german. 2013, pp. 31–39.

[10] KNX Association. "Systems Specifications, Communication Medium TP1".

[11] J. Postel. *Internet Protocol*. RFC 791. RFC Editor, 1981, p. 13. URL: https://tools.ietf.org/html/rfc791.

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

[13] Wolfgang Granzer et al. *Security in Networked Building Automation Systems*. Tech. rep. 2005.

[14] KNX Association. *Application Note 158 – KNX Data Security*. Status: Draft Proposal. May 2013.

[15] W. Granzer et al. "Securing IP backbones in building automation networks". In: *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009, pp. 410–415.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] R. R. Coveyou and R. D. Macpherson. “Fourier Analysis of Uniform Random Number Generators”. In: *J. ACM* 14.1 (Jan. 1967), pp. 100–119. ISSN: 0004-5411.