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)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

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 Motivation

Konnex (KNX) is an open communications protocol for Home and Building Automation System (HBA). It uses a layered structure and supports wired communication over twisted pair and power line as well as wireless communication by radio transmission. Additionally, it supports communication with Transport Control Protocol (TCP)/Internet Protocol (IP) hosts by special gateways. As such, it can be used for controlling traditional services like Heating, Ventilation and Air Conditioning (HVAC), but also for more sophisticated applications like surveillance or fire alarm systems of buildings [1].

Driven by the need to reduce maintenance costs and to improve usability, the application of HBAs is no longer limited to traditional HVAC services. Modern building management includes much more different and more sophisticated tasks like elevator control, alarm systems or access control, to name a few. To reduce costs it would be natural to bring together these services under the control of one management system, a claim also supported through improved standardization efforts.
Given these potential applications, a wide range of attacks would be possible. Replay attacks by intercepting and replaying datagrams would allow an adversary to introduce arbitrary KNX traffic, switching doors or disabling burglar alarms. Passive attackers can monitor the bus traffic to analyze the types of KNX devices within the network, gathering knowledge that can be used to develop further attack strategies. Denial of Service (DOS) attacks, disabling all directly connected KNX devices, can be conducted by simply physically shortcutting or interrupting a line connection, rendering the corresponding network segment unavailable. Clearly, such attacks must be precluded for sensitive services like fire or burglar alarms, relying on the availability of the communication network.

High availability, in general, can only be achieved by redundancy, i.e. by using replicated resources. Therefore, all resources needed for transmitting data between two points must exist redundantly and independently from each other.
The countermeasure against eavesdropping and replay attacks, providing integrity, confidentiality and authenticity, consists of authentication between the sender and receiver of a message, and encryption of these messages, combined in a security scheme called Authenticated Encryption (AE).

1.2 Problem statement

The origin KNX standard did not include countermeasures to prevent adversary misuse. This deficit was fixed afterwards by various extensions, bringing confidentiality and authenticity to KNX. Nevertheless, up to today no methods for increasing availability are provided, a fact considered problematic in view of the potential application domains. For example, a burglar can render an alarm system useless if he can shortcut the bus lines, thus suppressing the (encrypted) alarm messages. In contrast to such an malicious attack, a transient hardware failure can disable a system too, unacceptable for systems handling fire alarms or controlling elevators.

1.3 Aim of the work

The overall goal of this work is to develop a concept for a secure and highly available KNX network that also considers interoperability and compatibility, allowing the usage in environments even with increased safety-critical requirements. The proposed solution shall be resistant against malicious adversary as well as against transient hardware failures.
To achieve this, so called security gateways will be used. These gateways will possess two kinds of KNX interfaces: one kind of interface will be connected to standard, unsecured KNX networks. The second interface constitutes the entry point to a secured KNX network which is connected to the secure interfaces of other security gateways. To achieve higher availability, these secure interfaces and the respective communication lines must exist redundantly. This ensures that even in case of a DOS attack, communication within the segment is possible.

To show the feasibility of the solution by a proof of concept, a demonstration network shall be built. For the security gateways, RaspberryPis in combination with KNX-Universal Serial Bus (USB)-dongles will be used. Therefore, the RasperryPis are acting as gateways between the secure and the insecure KNX networks, each of them running a master daemon responsible for reading datagrams from the KNX insecure world, en-

crypting and authenticating them and sending them over the secure KNX lines.
It is important to note that the practical part of this work will only handle the twisted-pair media of KNX (i.e. KNX Twisted Pair (TP)-1), although the basic principles can be deployed in a modified manner in wireless and power line networks as well.
A threat analysis will be conducted to prove that the system can withstand the defined attacks and is robust, i.e. that it can recover from erroneous states. This will be done by exposing the demonstration network to the various defined attacks.

1.4 Structure of the work

Chapter 2 introduces the required prerequisites for providing integrity and confidentiality, handling symmetric and asymmetric ciphers. Chapter 3 explains the term availability and is concluded by state-of-the art technologies implementing highly-available communication networks.
Chapter 4 explains the structure of the KNX standard, and introduces the most important cryptographic extensions.
Chapter 5 suggests a KNX prototype, applicable in environments with increased availability needs.
Finally, chapter 6 explains the implementation of the prototype, evaluates it and discusses the results.

CHAPTER 2

Prerequisites

2.1 Information Security

Information security is the effort to protect information - be it in electronic or physical form - from adversary threats. The domain in which the data is to protected can be time, i.e. protecting saved information for later usage, or space, i.e. protect information transmitted physically from one user to another. To protect information, three key objectives must be met:

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. Integrity can be separated into data integrity and system integrity. While the first one assures that information is only modified in an authorized manner, the latter one demands that a system performs its intended function, free from manipulation.
Availability guarantees that the system works promptly and information, which is needed by an entity to provide some service, is accessible.

Because all three properties go hand-in-hand with each other they are also called the *"CIA - triad"* [2] - successfully attacking 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

three basic concepts are interleaved and building a system which honors only parts of them will most likely lead to an insecure system.
In addition to the three objectives listed above, additional security objectives are used frequently: **authenticity** is the property of being genuine and being able to be trusted, while **accountability** allows to trace the actions of an entity uniquely to that entity.

Threats

Threats form a *potential* violation of security - a violation must not need to occur for there to be a threat. Stallings [3] divides threat consequences into four categories:

- **Unauthorized disclosure** is a threat aimed against confidentiality. It occurs when an entity gains access to information for which it is not authorized.
- **Deception** threatens system integrity, and may result in tricking an authorized entity into accepting (adverbially) modified data.
- **Usurpation** opposes data integrity s.t. a system is controlled by an unauthorized entity.
- **Disruption** compromises the availability s.t. the system services do not operate correctly.

Attacks

Attacks are the manifestation of a security violation, exploited by an attacker. A classification of attacks against communication networks is given in Figure 2.1.

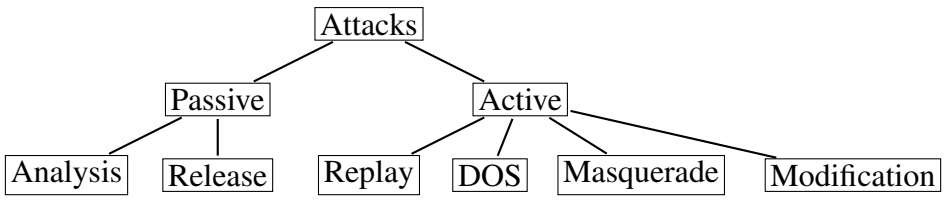


Figure 2.1: Classification of attacks

Passive attacks

Passive attacks are aimed against confidentiality. The attacker E intercepts the data transmitted between the two or more entities by monitoring the communication medium, recording all the traffic.

After obtaining the data, the attacker can **release** the data itself to an outside entity. Another way to exploit the data is to **analyze** them, allowing the attacker to get knowledge of some kind of meta data like origin, destination, quantity and frequency of the data flows.
Because of its passive nature the data sent are not modified, thus such an attack is hard to detect. On the other hand, encrypting the traffic to gain confidentiality suffices to guard against the release of data. To guard against traffic analysis is harder because it may be impossible to encrypt some kind of meta data: for example, the destination address of a message may not be encrypted because routers handling the message must be able to determine the next routing decision based on this address. Another reason is that some meta information will always be present, like the size of the message or the simple fact that communication between two points has occurred.

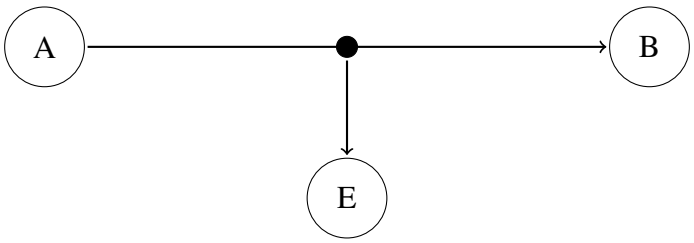


Figure 2.2: Passive attack

Active attacks

Unfortunately, a restriction to passive attacks only is no realistic assumption for many systems. Whenever the adversary has access to the communication medium, active attacks cannot be ruled out, allowing the attacker to modify the data stream, or to create false ones. In contrast to their passive counterparts, active attacks can be detected but not prevented, so the focus lies on detecting and recovering from active attacks.

A **replay** attack consists of two steps: first, the attacker monitors the traffic (i.e. conducts a passive attack) and injects this recorded package in the second step, thus trying to produce an unauthorized effect. Of course the package can also be modified and injected afterwards, resulting in a **modification** attack. A **Denial of Service (DOS)** attack tries to overload system resources, attacking availability s.t. the system is not usable by legitimate users. Finally, **masquerade** attacks occurs when one entity pretends to be another entity, often based on replay attacks by replying (modified) authentication messages of an authorized entity.

To prevent active attacks, integrity mechanisms must be combined with confidential-

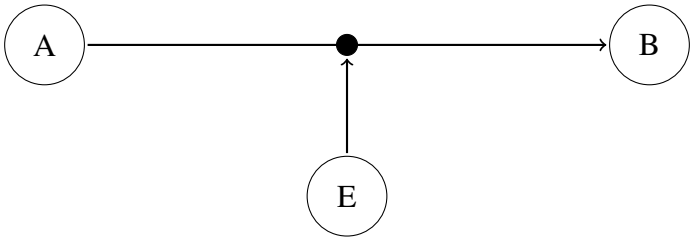


Figure 2.3: Active attack

ity mechanisms - the basic tool to achieve them is cryptography, introduced in the next section 2.2. To provide availability, different techniques must be used, as shown in Section 3.1.

2.2 Cryptography

Cryptography[1] is the science of encrypting information - its evolution was no linear process. Ciphers were used independently in different places, were forgotten and disappeared when the corresponding civilization died. A short time table for prominent events is presented below; for a comprehensive outline of cryptographic history, "The Codebreakers", written by David Kahn, is suggested [4].

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", named after its inventor and used 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 decryption 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

[1] classical greek for *kryptôs*: *concealed*
[2] classical greek for "riddle"

20th century introduces public key cryptography: 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 random looking stream or block of bits, called ciphertext, using some kind of *key*. This process is referred to as *encryption* in general, but it is important to note that for many block ciphers, this encryption process can also be used to generate a special tag called Message Authentication Code (MAC), providing integrity.
The next sections deal with how to achieve confidentiality, the concepts how to achieve integrity are partially based on the confidentiality methods and are introduced in section 2.3.

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
- $\mathcal{C}$ 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 $\mathcal{M} \to \mathcal{C}$. 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 $\mathcal{C} \to \mathcal{M}$, and is called *decryption function*.

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

The keys e and d are also referred to as *keypair*, written (e, d). If it is computationally easy to derive the private key e from the public key d (in most cases $e = d$), the encryption scheme is called *symmetric*, otherwise the scheme is called *asymmetric*.
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.

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

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

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

Security of a cipher

Perfect secrecy

A formal definition of a secure cipher was introduced by Shannon in 1949 [5], viewed from a communication-theory point of view, as follows: given a finite message space, every possible cleartext message has its own *a priori* probability (for example, the distribution of letters in a specific language). Additionally, every key can be chosen with specific probability. It is assumed that these two probabilities constitute the a priori knowledge of an attacker.
A message is picked, encrypted and sent to the receiver. The eavesdropper, intercepting the message, can calculate the *a posteriori* probabilities for all possible cleartext messages, leading to the observed cipher text - these are the conditional properties that under a fixed key, encrypting the cleartext message lead to the observed ciphertext message.
If the a posteriori probabilities for all possible encryptions are the same as all a priory probabilities, the attacker has learned absolutely nothing from intercepting the cipher text, which is defined by Shannon as *perfect secrecy*. Such a cipher cannot be broken by a ciphertext-only-attack, even by an adversary with unlimited time and processing power.
Shannon proofed that for a perfectly secure cipher, the key space must be 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 be different, allowing the attacker to get knowledge he should not have gotten.

[3] "runs in polynomial time"

Semantic security

Another definition of the security of a cipher, based on complexity theory, is *semantic security*. To be semantically secure, a cipher must not be breakable by an adversary in a reasonable time frame [6], where this time frame is a function of the useful timespan of the protected data. This synonymously means for a semantically secure cipher that an adversary must be forced to spend super-polynomial time to be able to break it. It follows that *all* semantically secure ciphers 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. Semantic security is therefore a weaker form of security, namely perfect secrecy against an adversary having only polynomially bounded processing powers [7].

Kerckhoff's principle

When designing cryptographic systems, 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.
The dutch cryptographer Auguste Kerckhoff stated rules for designing a secure cipher. According to *Kerckhoff's Principle* stated in the year 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. Shannon acknowledged these assumptions to be "pessimistic and hence safe, but in the long run realistic, since one must expect the system to be found out eventually".

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 be secret. The only thing that has to be kept private is the decryption key d. This separation of key and algorithm allows the publication of the basic cipher methods, benefiting from peer review. A contradicting approach trying to strengthen the safety of a cryptographic system by hiding the inner workings of it from public is also known as "Security by Obscurity".

Randomness and Probabilistic Theory

A basic requirement of all cryptographic schemes is the availability of randomness. *Entropy*, denoted H, is the unit of the unpredictability of a process, as defined by Shannon in [8]:

$$H(X) = -\sum_{x \in X} p(x) * ln(p(x)) \tag{2.1}$$

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 has a probability of $\frac{1}{2}$, and all flips are independent from each other [9]. If obtaining heads of the coin is viewed as a logical "0" and tails as a logical "1", a binary string of length n can be built, where the probability of all possible strings of same length is equal, as shown in Figure 2.4, yielding an *uniform distribution*, with $H = 2^n * \frac{1}{2^n} * ln(2^n) = n$ bit.

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. Also, many key negotiation algorithms schemes rely on determining a random prime number, which is often achieved by choosing a random number and testing it for primality. 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 Pseudo Random Number Generator (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 [10], which can in principle generate random numbers of sufficient quality, **if** the parameters are chosen correctly. 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, a fact that can be seen in Figure 2.5. 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

[4] John von Neumann, 1951

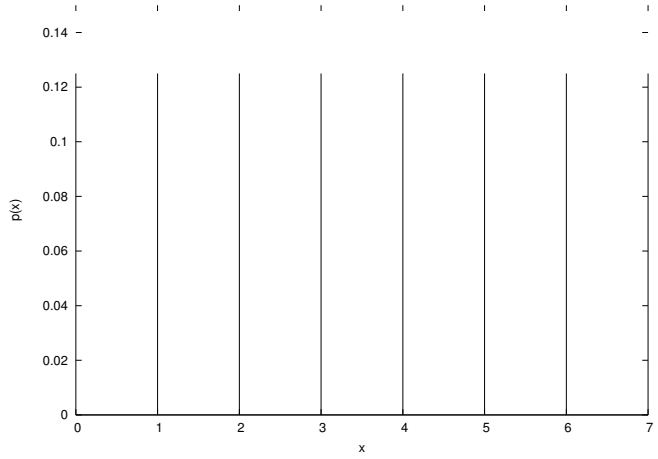


Figure 2.4: Uniform Distribution of binary string of length three

available, see [11] 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.
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 FIXME for the results. FIXME - DO IT?

The discrete logarithm problem

The Discrete Logarithm Problem (DLP), a mathematical problem, is the basis of two important key exchange algorithms, both introduced in section 2.5: the first one, Diffie-Hellman (D-H) utilizes finite fields, whose theoretical background is introduced below. Elliptic Curves (ECs), basis for the second algorithm, are also explained.

Finite Fields DLP

A field consists of a set $\mathcal{F}$ together with two operations $\cdot$, namely addition "+" and multiplication "*", satisfying the following properties:

- Closure: for all elements from $\mathcal{F}$, the set is closed under the defined operations, i.e. applying an operator $\cdot$ to two elements from the set results in an element also belonging to the set.
- Associativity and commutativity hold, i.e. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ and $a \cdot b = b \cdot a$

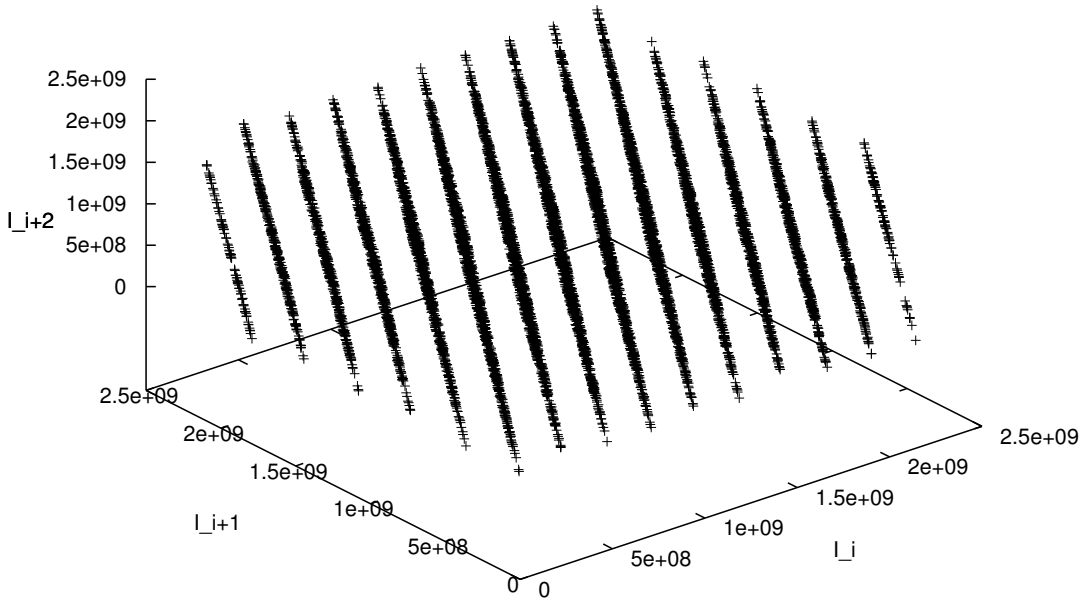


Figure 2.5: Spectral Plot of RANDU output

- For both operations and all elements from the set $\mathcal{F}$ an identity element e exists s.t. $a \cdot e = a$
- For both operations and all elements from the set $\mathcal{F} \backslash \{0\}$ an inverse element exists s.t. $a \cdot a^{-1} = e$
- Distributivity holds: $(a + b) \cdot c = a \cdot c + b \cdot c$ for all elements

For a finite field, the size of the set $\mathcal{F}$ is finite and called *order* of the field, with neutral elements for addition and multiplication. Inverses can be found to for all elements regarding addition s.t. $a + (-a) = e$ and regarding multiplication for all elements except 0 s.t. $a * a^{-1} = e$.

A specific example of a finite field is a prime file, which can be constructed by taking the set of integers $\mathbb{Z} \pmod p$, with $p \in \mathbb{P}$, thus restricting the set of all integers to the set $\mathbb{Z}_p = \{0, 1, ..., p - 1\}$. By choosing p as prime it is ensured that for any element $a \in \mathcal{F} \backslash \{0\}$, a multiplicative inverse exists: $a * a^{-1} = 1 \pmod p$. The set

of numbers for which multiplicative inverses exist is called ${\mathbb{Z}_p}^*$, so for p being prime, $\mathbb{Z}_p \backslash \{0\} = {\mathbb{Z}_p}^*$.
By raising an element $a \in \mathbb{Z}_p$ to different powers, a subgroup of ${\mathbb{Z}_p}^*$ is generated, a fact that follows from Fermat's little theorem stating that for p being prime and raising a to the power $p-1$, the outcome is congruent 1 modulo p:

$$a^{p-1} \equiv 1 \pmod{p}$$

Raising a to higher powers results in:

$$a^p \equiv a * a^{p-1} \equiv a \pmod{p}$$
$$\dots$$
$$a^{2p} \equiv a^2 * a^{p-1} * a^{p-1} \equiv a^2 \pmod{p}$$

Thus, generating higher powers than $(p-1)$ does not yield different outcomes. If by raising a to $(p-1)$ different powers all elements from ${\mathbb{Z}_p}^*$ can be generated, a is called *primitive root* or *generator*, generating a cyclic group. If p is prime it can be shown that at least one such generator g must exist and can be found efficiently [12]. Conversely this means that for all elements from ${\mathbb{Z}_p}^*$ a unique exponent in the interval $[0, p-1]$ exists, called the *discrete logarithm for the base* $g \pmod{p}$.

To find the discrete logarithm, several algorithms exist. The most naive method conducts an exhaustive search, so for a prime of n bits length, 2^n search operations are necessary. More efficient algorithms exist, with the best methods finishing after about $2^{\frac{n}{2}}$ steps [13]. Therefore, by choosing a sufficient large prime, finding the discrete logarithm is considered a hard problem, a fact that is exploited by the D-H key exchange algorithm and its variants.

Elliptic curve DLP

An EC is basically the set of all points satisfying an equation with the form as shown in 2.2, called "Weierstraß" equation:

$$y^2 = x^3 + ax + b \tag{2.2}$$

The additional condition $4a^3 + 27b^3 \neq 0$ assures that the EC does not possess any singularities.
An imaginary point "in infinity", denoted ∞, serves as additive neutral element, and also belongs to the EC by definition:

$$P + \infty = \infty + P = P, P + (-P) = \infty$$

Negatives can be calculated easily due to the symmetric nature of the curves by swapping the sign of the y-coordinate of the point. For point P with coordinates (x, y), $-P$ is defined as $(x, -y)$, thus satisfying $P + (-P) = \infty$.
By defining an EC over $\mathbb{Z}_p$, with $a, b \in \mathbb{Z}_p$, a cyclic group can be generated. The addition operation that "adds" two points on this curve is defined by a "chord-and-tangent" rule. Figure 2.6 shows addition of two points, while 2.7 shows adding a point to itself. It is important to note that this representation is shown in the domain $\mathbb{R}$ for visualization - in real EC cryptography, only elements $\pmod p$ (i.e. from the set $\mathbb{Z}_p$) are allowed to avoid rounding errors.

Addition is defined as connecting points A and Q, finding the intersection of the line with the EC (R') and reflecting that point across the x-axis to obtain the result R.

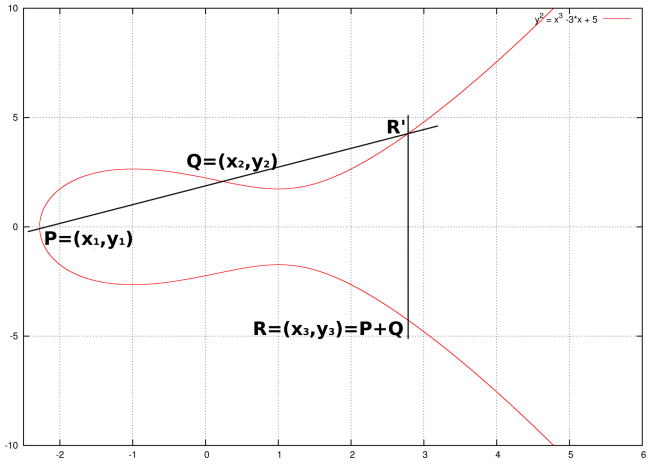


Figure 2.6: Adding two points

Mathematically, the new coordinates can be calculated as shown in equation 2.3:

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2, y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1 \tag{2.3}$$

Point-doubling of P is achieved in a similar way by determining the tangent of P, finding the intersection and reflection.

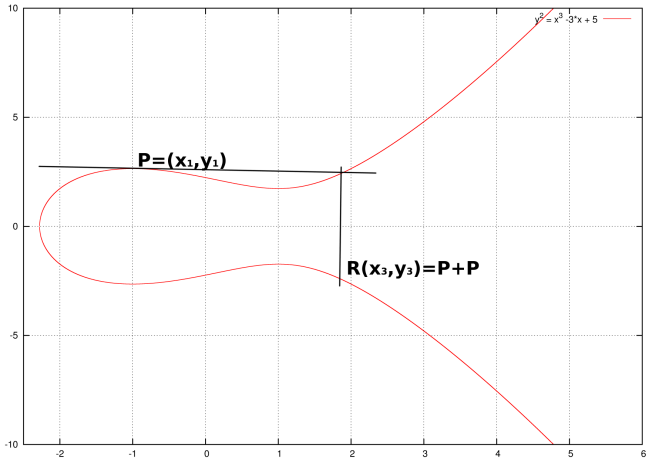


Figure 2.7: Doubling a point

Multiplication is defined as successive addition of a point to it self in the way

$$kP = \underbrace{P + P + ... + P}_{k-times}$$

Elliptic Curve Discrete Logarithm Problem (ECDLP)

Based on this group operations, the ECDLP [14] can be defined as following: given an EC E over the finite field $\mathcal{F}_p$, P of order n of this curve and a point Q, find the integer k s.t. $Q = kP$. k is called the discrete logarithm of Q to the base P.

To find k, the most naive algorithm trying all possible numbers will terminate after $\frac{n}{2}$ on average. In contrast, the running time of the best known attack is bounded by $\sqrt{p}$, where p is the largest prime divisor of n. Therefore, such an attack is infeasible for properly chosen EC parameters.

2.3 Symmetric vs. Asymmetric Cryptography

As already stated, two very fundamental differences regarding the key used in a cryptographic system can be found. Symmetric ciphers, where the same key is used for encryption and decryption, outperform its asymmetric counterparts in regards of data throughput by a factor of about 1000 [15]. Additionally, they need shorter keys to achieve the same level of security - both arguments encourage its 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 for key exchange. But if such a secure channel can be established, it could also be used for transmitting the sensitive data themself.
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 both schemes, a hybrid approach 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

Most stream ciphers belong to the family of symmetric ciphers, thus $e_i = d_i$. The reason is that most asymmetric ciphers are deterministic ciphers, i.e. the encryption of the same message with a fixed public key always yields the same cipher text. Thus, such repeated messages can be detected by an adversary. Probabilistic public-key encryption can solve this problem for stream ciphers, but this scheme will not be handled in this work because of its low practical application.
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 Exclusive-Or (XOR) operation, as defined in table 2.3, of key and message.

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

Decryption can be achieved by applying the XOR operation to key and ciphertext:

$$m_i = c_i \bigoplus k_i = (m_i \bigoplus k_i) \bigoplus k_i = m_i, \text{ with } k_i \bigoplus k_i = 0, const \bigoplus 0 = const$$

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: for a n-character ciphertext, **all** n-character cleartexts are equally probable, and vice versa. The reason for this is the XOR operation: both outcomes are equally probable, introducing one bit of randomness into every data bit.
Additionally, the XOR operation can be built easily in hardware, accelerating the encryption or decryption process.
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. If an attacker is able to intercept a high number of different ciphertexts, all encrypted with the same key, the 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 with high probability 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 shorter, initial key, called *seed*. Such LFSR are denoted by the tuple $\langle L, C(D) \rangle$. L is the number of stages, and $C(D)$ is the *connection polynomial*. Because of the finite length, every LFSR can only take on a finite number of internal states, producing a periodic output sequence. If the degree of the connection polynomial is equal to the number of stages and the connection polynomial is irreducible (i.e. the polynomial can not be factored into 2 non-constant polynomials), no matter of the initial state, the output sequence produced will always be of maximum periodicity.
Figure 2.8 shows a 4 stage non-singular LFSR with

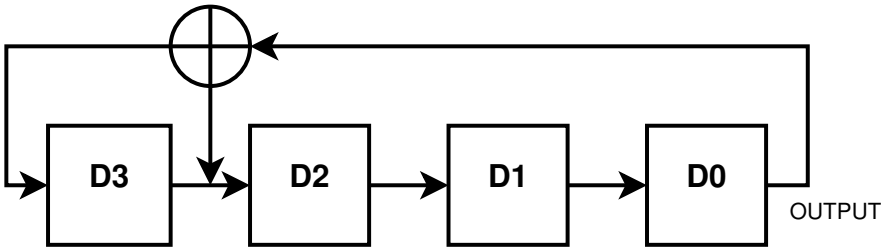


Figure 2.8: 4 Stage LFSR

$$L = 4, C(D) = 1 + D + D^4,$$

Table 2.3 [16] shows the corresponding output sequence produced. After 15 shifts a state equal to the initial state is achieved, and the outputs begin to repeat.
While such LFSR can be easily built in hardware, a problematic fact remains that their *linear complexity* is bounded by L. Therefore, a LFSR should never be used as keystream generator directly, instead the outputs of different LFSR are combined by a non-linear function, thus obtaining a nonlinear generator.

t	D_3	D_2	D_1	D_0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0

t	D_3	D_2	D_1	D_0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	0	1	1	0

Block Ciphers

These ciphers operate on input blocks of fixed size, transforming them into output blocks of same size. This implies that larger messages must be broken into suitable blocks, and that for the last remaining block it may be necessary to add padding bytes to yield the full block size, adding overhead to the message - a disadvantage compared to stream ciphers. For example, to encrypt a message just exceeding the block size by one byte, for the excess byte a complete block must be concatenated.
On the other hand, while stream ciphers are strictly sequential by nature, there exist methods to speed up block ciphers by splitting the message first, and then process them in parallel[5].

[5]Counter Mode, see 2.3

Two main types of block cipher exist: *transposition* ciphers use a key-dependent permutation to re-order the characters of the block to obtain the ciphertext. This is a bijective transformation, so decryption can be achieved by simply reversing the permutation.
Substitution ciphers define a key-dependent mapping of characters from the alphabet $\mathcal{A}$ to the same alphabet, thus replacing every character by one or more other characters. In the latter case, this equals an injective function which can not be reversed directly.
A product cipher is a combination of ciphers of different types to achieve a higher level of security than possible as with the basic ciphers.
Feistel networks are special product ciphers, composed of Substitution-Permutation (SP) networks. They were first described by Horst Feistel in the year 1973[17], and are the basis of a variety of block ciphers like "LUCIFER" [18], developed by Feistel, and Data Encryption Standard (DES).
Figure 2.9 shows the principal layout of such ciphers: at first, the plaintext block of length 2n-bits is divided into two n-bits blocks, often called L_0 and R_0 for left and right block, respectively. After that the first round starts: every round is characterized by performing a substitution, followed by a permutation of the two half-blocks. For substitution, at first a *round function*, parametrized by a *round key* is applied to one half of the data block, followed by a XOR operation. The output of the rounds can be calculated according to the formulas shown in 2.3

Encryption of round 1:	$L_1 = R_0$
	$R_1 = L_0 \bigoplus F(k_1, R_0)$
Encryption of round 2:	$L_2 = R_1$
	$R_2 = L_1 \bigoplus F(k_2, R_1)$
...	
Encryption of round n:	$L_n = R_{n-1}$
	$R_n = L_{n-1} \bigoplus F(k_n, R_{n-1})$

Decryption is achieved by applying the ciphertext to the same network, with the round keys applied in reverse order, reducing hardware- respectively code size. Because every decryption step, see 2.3, does not rely on reversing the round function, there is no necessity for the round function to be bijective.

Decryption of round n:	$R_{n-1} = L_n$
	$L_{n-1} = R_n \bigoplus F(k_n, R_{n_1}) = R_n \bigoplus F(k_n, L_n)$

DES and Triple Data Encryption Standard (3DES)

DES, designed by IBM and published by National Institute of Standards and Technology (NIST) in 1977 [19], encrypts 64 bit blocks in 16 processing rounds.

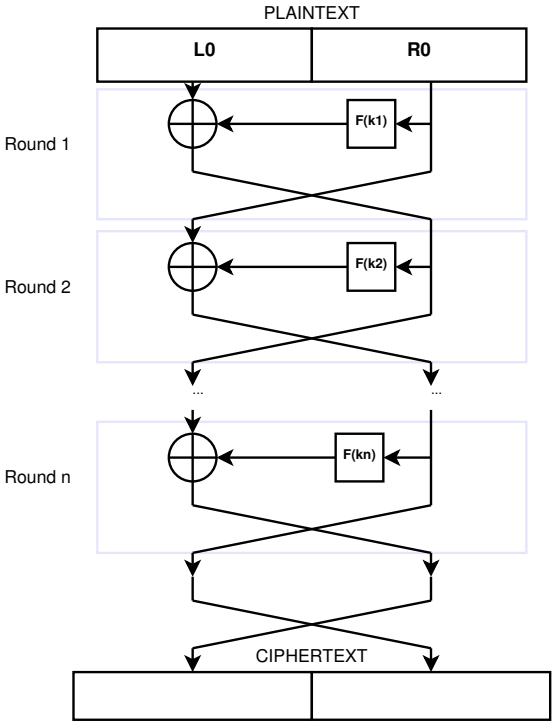


Figure 2.9: Feistel Substitution-Permutation Network

For every round, a 56 bit round key is derived from the basic 56 bit key by permutations. The 64 bit data block to be encrypted respectively decrypted is subjected to an initial permutation and then feed into the Feistel network. The round function operates as follows:
At first, the 32 bit half block is expanded to 48 bit by copying specific bits. The outcome is added to the round key modulo 2 (i.e., the XOR operation). Next, a non-linear transformation is applied by so-called "S-Boxes", performing a surjective function by substituting blocks of 6 bit by only 4 bit. Lastly, a deterministic permutation follows, achieved through "P-Boxes", concluding the round function.
Because of the small key size, DES was successfully broken for the first time[6] by a brute-force attack in 1997.
To prevent such attacks, 3DES was published: the cleartext- respectively cipertext block is feed 3 times to the DES cipher, using 3 different keys k_1, k_2, k_3 to first encrypt with k_1, decrypt with k_2 and finally encrypt with k_3, effectively tripling the key size:

$$ciphertext = E(k_3(D(k_2, E(k_1, cleartext)))$$

[6] At least officially - rumors about the involvement of the National Security Agency (NSA) regarding the small key size and the design of the S-Boxes existed since the publication

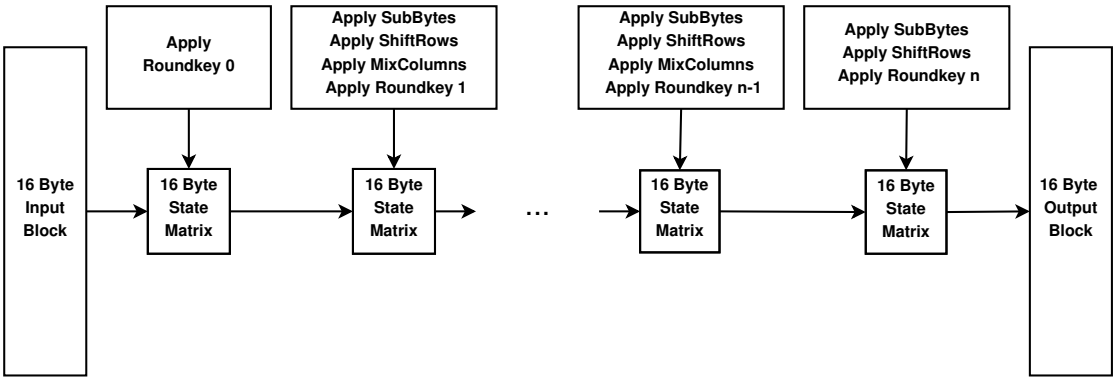


Figure 2.10: AES Encryption Process

The special sequence of encryption, decryption and again encrypting was chosen because by setting $k_1 = k_2 = k_3$, a 3DES implementation can also be used for en/decryption of DES messages.

Advanced Encryption Standard (AES)

Also called "Rijndael" after its developers Joan Daemen und Vincent Rijmen, AES is is the successor of 3DES, as proposed by the NIST in 2001. Basic properties are a block size of 128 bit, and possible key sizes of 128, 192 or 256 bit.
The operation, shown in Figure 2.10, starts by copying the input block into a square matrix, called "State", followed by a XOR combination of the first round key and the matrix. Then, 9, 11 or 13 rounds, depending on the key size, are performed: substitution by S-Boxes, permutation by shifting rows, another substitution by mixing columns and applying the round key. A last round, omitting the mix-columns stage, concludes the encryption. Operating on the whole data block, AES is not a Feistel network, therefore all substitutions and permutations must be reversible to allow decryption: the S-Box used here is therefore implementing byte-by-byte substitutions. The round keys are derived from the origin key by the AES key expansion. Decryption uses the round keys in reverse order. To reverse the first substitution of every round, a unique inverse S-Box is used, while the shifting rows and mixing columns can also be reversed.

Mode of Operation

Because block ciphers operate on a fixed number of bytes, messages larger than this block size must be broken into parts of suitable size, and depending on the resulting size of the last block, it may be necessary to append a padding to it. Five different modes of operation were defined by NIST in 2001 [20], which will be introduced in the

next sections. For all modes it does not matter what underlying block cipher is used, as long as the block cipher implements a cryptographic secure function.
An important property of this modes is the error propagation. Whenever a bit error occurs on the transmission channel due to noise or interference, a logical '0' of the transmitted cipher text is substituted by a logical '1' or vice versa. This bit error in the cipher text produces one or more bit errors in the clear text, thus the name error propagation [21].

Electronic Code Book (ECB)

ECB can be used to gain confidentiality and allows the parallel processing of all input blocks. This mode does not use any IV or nonce[7], therefore repeating input blocks are mapped to the same output blocks under the same key, implementing a deterministic encryption scheme. This is problematic, which can be seen quite intuitively by comparing Figures 2.11 and 2.12. Therefore, this mode should never be used.

Figure 2.11: Unencrypted Picture

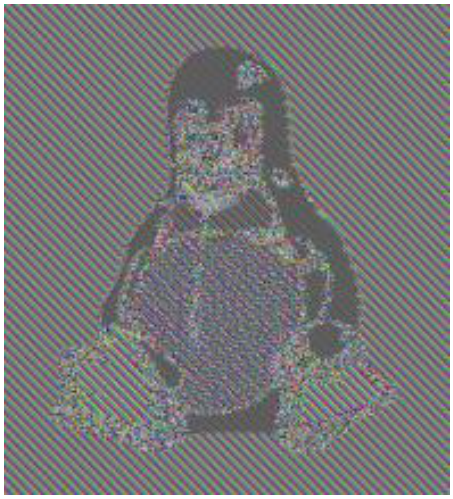


Figure 2.12: ECB encryption of the picture

Cipher Block Chaining (CBC)

This mode uses an IV and can therefore be used for encryption of same messages without changing the key. Additionally, CBC can also be used for MAC generation, as shown in section 2.3.
Encrypting a message is shown in Figure 2.13.

[7] short for "*n*umber used *once*"

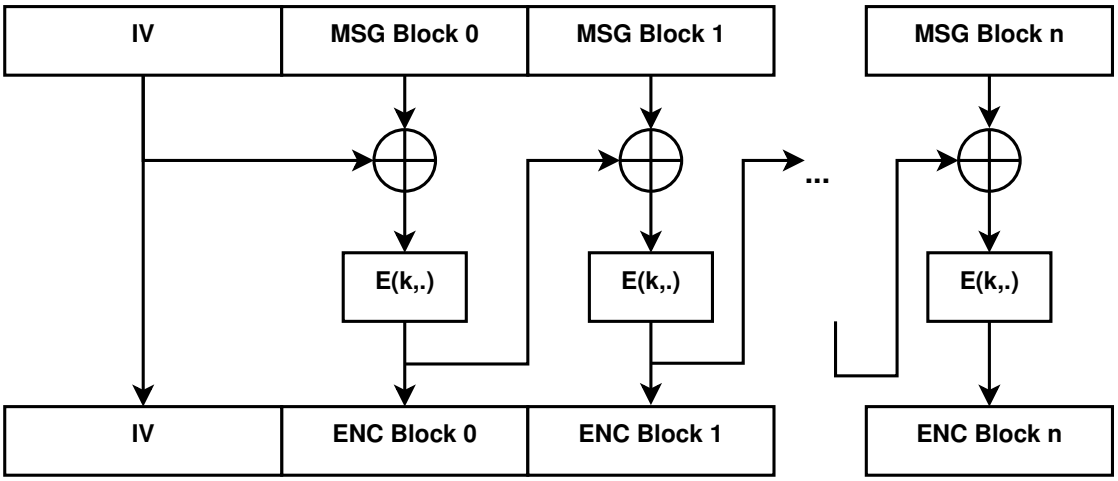


Figure 2.13: CBC for encrypting messages

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

To reverse the process, i.e. decrypt the message, see Figure 2.13

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

The IV does not have to be kept private, but must be known to the receiver of the message. It is important that such an IV is unpredictable, otherwise allowing a Chosen Plaintext Attack (CPA). Also, it must not repeat over the lifetime of the key, otherwise introducing the ECB problem again.
The IV introduces overhead, which is more problematic for shorter messages. To avoid such a message expansion, a solution is to use a nonce, as suggested in [22]. Sender and receiver must maintain a message counter. This message counter must be encrypted to avoid predictability, and can then be used as IV. Care must be taken for the counter not to overflow within the lifetime of a key.

Counter Mode (CTR) mode

This confidentiality mode generates a key stream by encrypting a counter value with a block cipher. The key stream is then applied to the cleartext blocks with the XOR

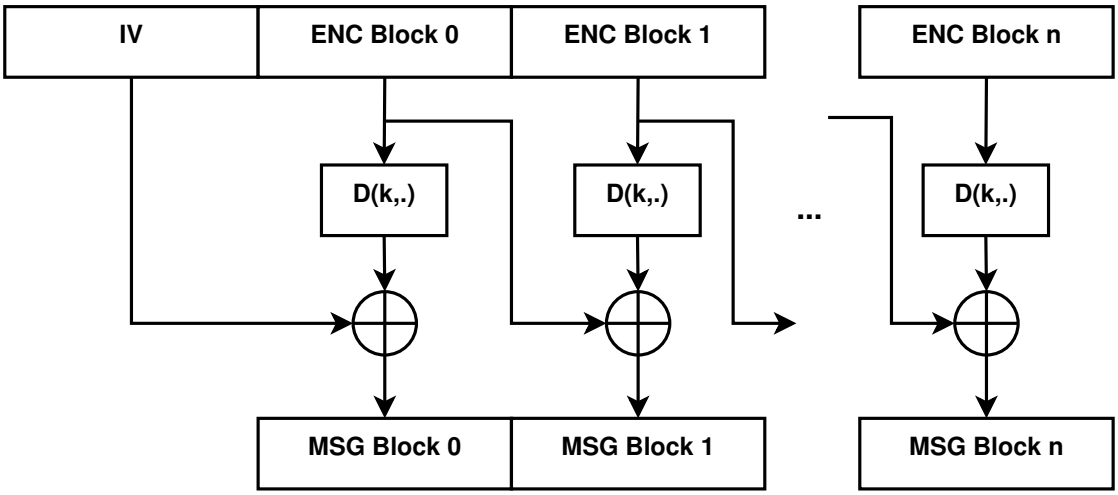


Figure 2.14: CBC for decrypting messages

operation, as shown in Figure 2.15. For the last block, the key stream is truncated to match the size of the cleartext block.
Decryption works by generating the same key stream on the receiver's side, and applying the XOR operation to the ciphertext blocks, similar to the decryption process used in the Vernam cipher. To avoid the duplicate usage of the same counter value in bidirectional communication, the counter can be combined with a sender-dependent nonce by concatenation before encrypting the counter:

$$K_0 = E(k, nonce||Ctr_0)$$
$$K_1 = E(k, nonce||Ctr_1)$$
$$...$$
$$K_i = E(k, nonce||Ctr_i)$$
$$C_i = K_i \bigoplus M_i$$

Cipher Feedback Mode (CFB)

CFB can be used to turn a block cipher into a stream cipher. Beside the block size b, another parameter s determines the operation. s corresponds to the size of one transmission unit. For initialization, an unpredictable IV is set as input for the underlying block cipher. Then, in every processing step a new transmission unit is generated by XORing the s most significant bits of the output of the encryption function with the s bit message unit. After that, the IV is shifted to the left and the gap is filled by the newly generated character, as shown in Figure 2.16:

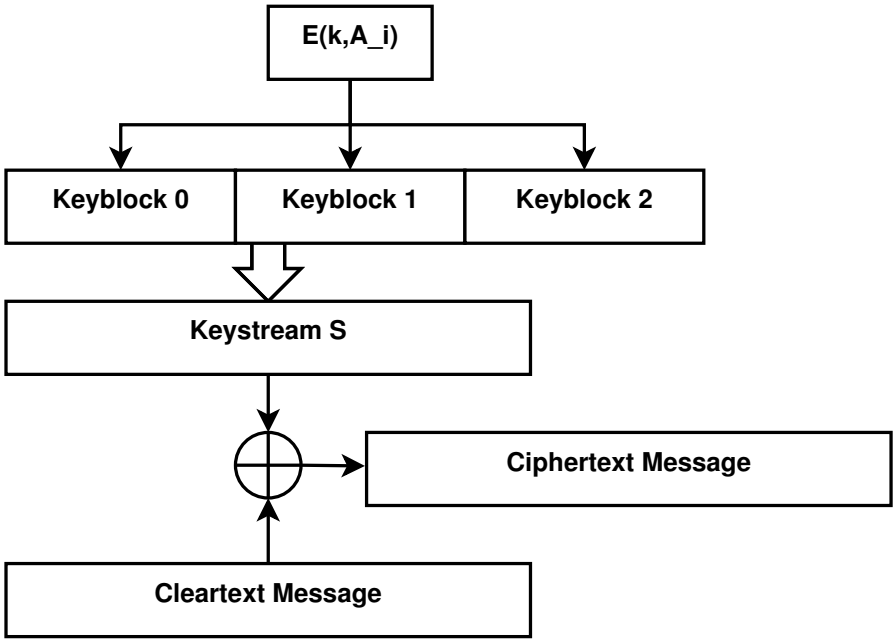


Figure 2.15: CTR mode encryption

$C[0] = E(k, IV[0 : b-1])[b-1 : b-s-1] \bigoplus M[0]$
$C[1] = E(k, IV[0 : b-s-1]||C[0])[b-1 : b-s-1] \bigoplus M[1]$
...
$C[n] = E(k, IV[0 : b-ns-1]||C[0]||C[1]||..||C[n-1])[b-1 : b-s-1] \bigoplus M[n-1]$

To decrypt, the same encryption function, IV and key is used to retrieve one transmission unit at a time.

Output Feedback Mode (OFB)

This mode is very similar to CFB, but here the s bit output from the encryption function is used directly to update the space caused by the IV left shift. This avoids error propagation in case a transmission unit was damaged on transmit and thus a bit changed its value: for OFB encryption systems, one or more bit errors in one ciphertext character only affects the decryption of this character. In contrast, one bit error in CFB affects decryption of all following characters.

$O_0 = E(k, IV[0 : b-1])[0 : s-1]$
$C[0] = O_0 \bigoplus M[0]$
$O_1 = E(k, IV[0 : b-1]||O_0)[0 : s-1]$

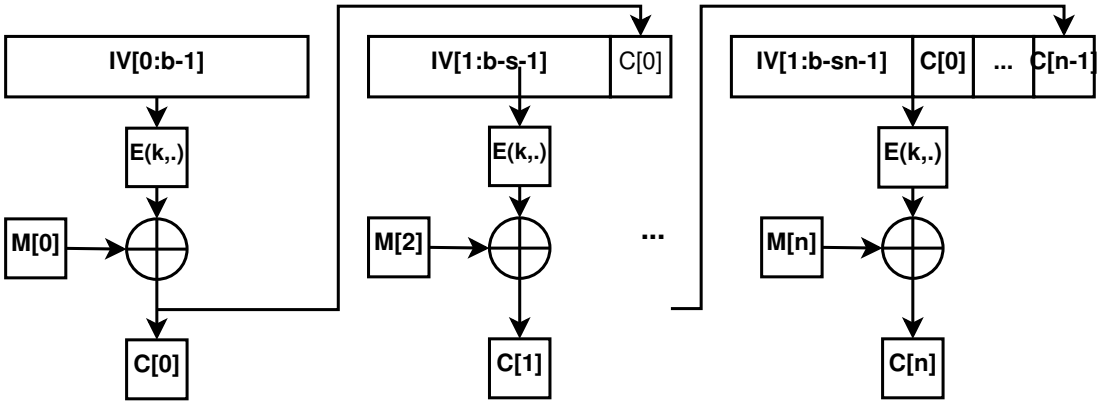


Figure 2.16: CFB Encryption

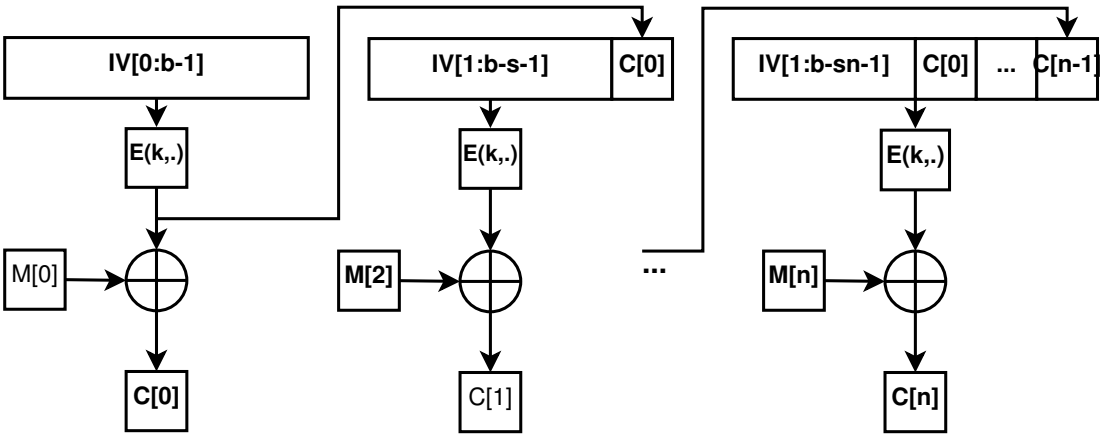


Figure 2.17: OFB Encryption

$C[1] = O_1 \bigoplus M[1]$
...

$0_n = E(k, IV[0 : ns - 1]||O_0||O_1||O_{n-1})[0 : s - 1]$
$C[n] = O_n \bigoplus M[n]$

Integrity

All modes of operation introduced so far are used to provide confidentiality only. To provide the second column of information security, i.e. integrity, in general MACs or

digital signatures are used. Digital signatures, based on public key cryptography, are introduced in Section 2.5. In contrast, MACs use symmetric keys, providing integrity and authenticity.
The basic way to protect a data unit - be it a message traveling from sender to recipient or a file saved for later usage - from modification by a third party is to generate a tag t, also called MAC, and concatenating this tag to the message, as shown in Figure 2.18. Verifying the integrity is done by re-generating the tag and comparing it to the saved one.

In contrast to confidentiality modes, which should always be combined with a method to guarantee integrity, integrity-only modes do have a right to exist. For example, archives containing source code for open source project, available from the internet, must not be encrypted but should be secured against modification. For example, the UNIX based operating system "FreeBSD" uses asymmetric keys to protect its package management system from adversary modification.

Combining integrity and confidentiality in a security scheme called AE will be handled in section 2.4.

As tag generation and tag verification algorithms, keyed or unkeyed cryptographic secure hash functions are used. For integrity-only, keyed hash functions must be used, or otherwise an arbitrary entity can modify the message undetectable by regenerating the correct tag. Therefore, a simple checksum like a Cyclic Redundancy Check (CRC) or an unkeyed hash function can not provide integrity only, simply because the function value can be re-generated by an adversary modifying the message, allowing to mount an attack called MAC forgery. Nevertheless, in combination with a confidentiality mode such an unkeyed function *may* be secure, although its use is discouraged because many applications operating this way are broken and can be attacked (i.e. Secure Shell (SSH) version 1 [23]).

A hash functions takes as input an arbitrary large message $\mathcal{M}$ and generates a hash value $t = h(\mathcal{M})$ of fixed size, therefore $|M| >> |t|$. This many-to-one mapping implies the existence of collisions, i.e. the existence of distinct messages m_1, m_2 which are mapped to the same tag t.

To be cryptographically secure, a hash function must fulfill specific properties [24]: firstly, while it should be easy to generate the tag by calculating $t = h(m)$, reversing the process to get m by executing $h(t)^{-1}$ should be hard, a property called *preimage resistance*. Additionally, *2nd-preimage resistance* assures that for any given message m and corresponding tag t, it must be infeasible to find a second message that maps to the same tag. Finally, *(strong) collision resistance* states that it also must be infeasible

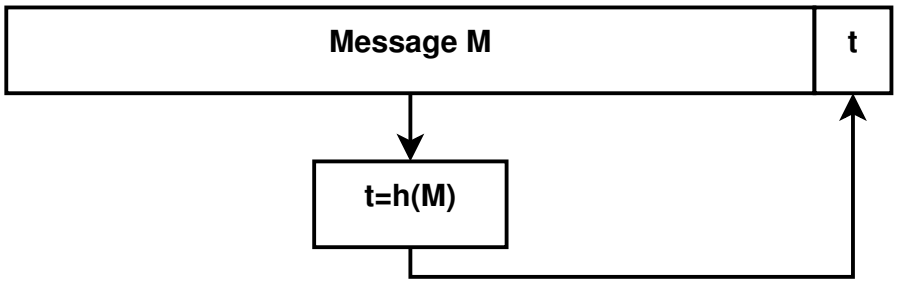


Figure 2.18: Tag generation

to find any two messages generating the same tag, therefore collision resistance implies 2nd-preimage resistance [25].

Important representatives of cryptographically secure hash functions are the family of Standard Hashing Algorithms (SHAs), with variants providing hashes of 160, 256, 384 or 512 bit length, defined by NIST [26]. For SHA-1, an attack to find collisions was found [27] and should therefor not be used anymore.
Because these hash functions lack a secret key and therefore would allow MAC forgery, a construction called Keyed-Hash Message Authentication Code (HMAC) can be used, which takes additionally to the message m as input a key k, and $ipad$ and $opad$ being constant values [28].

$$t = S(k, m) = h((k \bigoplus opad)||h((k \bigoplus ipad)||m)) \tag{2.4}$$

A different way for tag-generation, called CBC-MAC, is based on block ciphers, utilizing a construction similar to CBC mode encryption, shown in Figure 2.19.

2.4 Authenticated Encryption

Many applications demand the integration of confidentiality *and* integrity modes. The need for confidentiality is self explaining in systems where submitted data must be protected from a passive adversary. One example would be a password for logging into a remote computer, sent over a network connection - just by monitoring the connection, an attacker can steal the password and gain illegitimate access to the system.

The need to additionally provide integrity and authenticity measurements is maybe not that obvious, but can be motivated in various ways. For example, two entities sharing

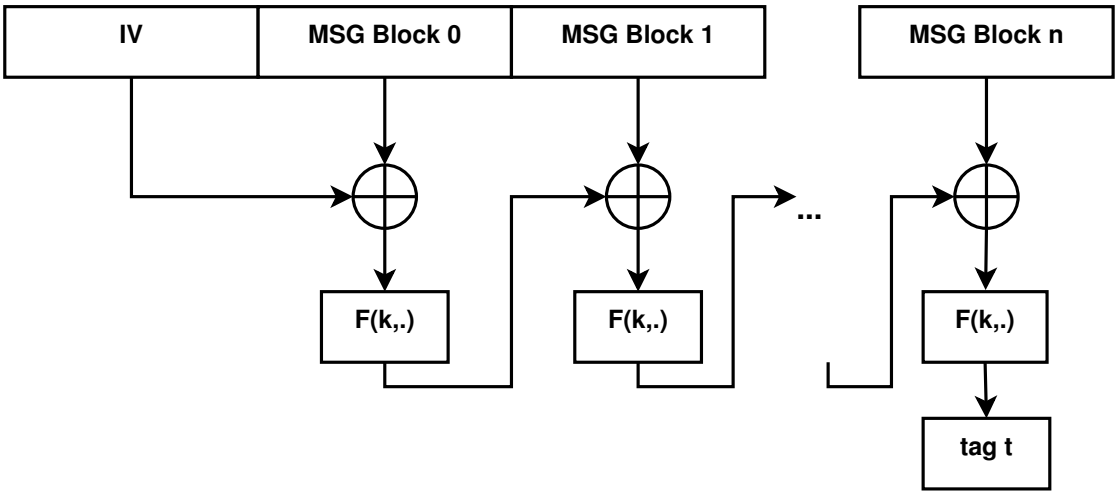


Figure 2.19: CBC for generating a MAC

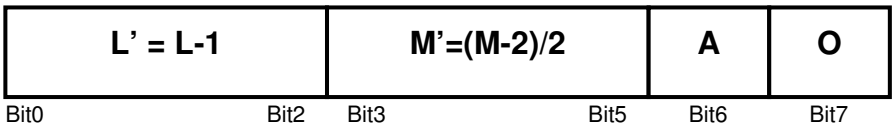


Figure 2.20: Flag Field of CBC IV

an initial key known to both of them which want to negotiate a temporary session key could randomly choose the temporary key, encrypt it and send it to the other side. Even if a passive adversary knows that the package he monitors will be used as session key he cannot derive it, provided a secure cipher is used. In contrast, an active attacker can intercept and modify the encrypted session key, and afterwards inject the modified message. The receiver would then decrypt the package, thus obtaining a modified session key. After all, the two entities would use two different keeps, crippling their communication.

To avoid that attacks, methods to provide encryption and integrity are combined. Basically, 3 different ways how this can be achieved exist:

- "encrypt-and-mac" encrypts the message m and appends the tag t of the cleartext message

$$c = E(k_e, m)||h(k_h, m) \tag{2.5}$$

- "mac-then-encrypt" generates the tag for the cleartext message m, appends it and

afterwards encrypts cleartext + tag to get ciphertext c:

$$c = E(k_e, m||h(k_h, m)) \tag{2.6}$$

- "encrypt-then-mac" encrypts the message m first, and afterwards appends the tag t of the encrypted message to obtain c

$$c = E(k_e, m)||h(k_h, E(k_e, m)) \tag{2.7}$$

For all 3 schemes, $E(k, m)$ must be a semantically secure encryption function, and $h(k, m)$ denotes a keyed, cryptographically secure hash function. The latter is of particular importance for 2.5, because this scheme otherwise directly leaks information about the plaintext in the tag.
As general, cryptographic data should always be authenticated first. Only if authenticity can be verified, the encrypted data should be processed further. Following this rule, it turns out that only 2.7 is considered generically secure against chosen plaintext attacks. "Encrypt-and-mac" [29], used in SSH, is considered generically insecure. For "mac-then-encrypt", used in Secure Sockets Layer (SSL), the same holds true, but this scheme can be used in a secure manner if CBC or a stream cipher like CTR mode is used for encryption.

CCM

Counter with CBC MAC (CCM) [8], combines CBC for authentication and CTR for encryption. CCM generates the MAC for the message first, appends this MAC to the cleartext data and afterwards encrypts data and 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 Medium Access (MAC), so valid values are $M \in \{4, 6, 8, 10, 12, 14, 16\}$.

- 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 2.19.

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

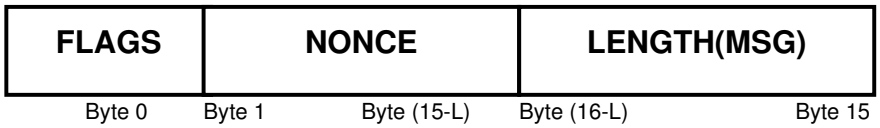


Figure 2.21: IV for CBC MAC

Both parameters are encoded in the very first byte of the first message block, thus reducing the possible maximum size of the nounce, as shown in Figure 2.20. Bit 6 of the length field is set to 1 if additional authenticated data are sent, and bit 7 is reserved and set to 0.

Generating the MAC

As shown in Figure 2.19, the first message block M_0 is XOR'd with a nonce or IV (see Figure 2.21), which must be unique per key to avoid deterministic encryption. The result of the XOR operation is then feed to the block-cipher to get the first cipherblock C_0. The encrypted data C_0 gets XOR'd with the next message block M_1, and this result becomes the input for the block cipher, and so on, iterating over all n message blocks to determine the tag t:

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

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

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

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

Encrypting Data and MAC

CTR 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 (generating the MAC first, then encrypt data and MAC) *may* be vulnerable to padding oracle attacks [30], CTR effectively avoids these attacks simply because there is no padding needed.
CTR implements a weaker form of the one time pad by generating a keystream of sufficient length, and then applying the XOR operation to the keystream and the data, as

shown in Figure 2.15.

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 (i.e., the length of the message to encrypt). This obtained keystream is then bitwise XOR'ed with the cleartextmessage (which consists of the data and the MAC), yielding the final encryption.

2.5 Public Key Cryptography

Public Key Cryptography solves the problem of establishing a secure channel by using an insecure one. Here sender and recipient 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 fundamental 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. The basic idea for such a one-way function was formulated for the first time in the year 1874 by William Stanley Jevons stating:

"Can the reader say what two numbers multiplied together will produce the number 8616460799? I think it unlikely that anyone but myself will ever know." [31]

Although his statement was not related to cryptography at all, and of course factoring of much bigger numbers is doable nowadays, this statement exactly describes the spirit of public key systems, and the security of RSA, introduced below, is directly connected to the inability to factor large numbers in reasonable time.
Because disclosure of the public key does not affect the security of the scheme, the public key can be published in some sort of dictionary. 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 its participants, or it will be vulnerable to man-in-the-middle attacks.

Merkle Puzzles

In [32], Ralph C. Merkle developed an algorithm for key exchange between two parties. While the algorithm is based on symmetric ciphers and is not practicable, it motivates the usage of public key systems based on algebraic structures and is therefore introduced.

The key idea is that the necessary work by the two legitimate parties when negotiating a key is bounded by $O(n)$, while an adversary must spend $O(n^2)$ to also calculate the key, thus generating a quadratic gap. Merkle defines a puzzle as cipher text that is supposed to be broken. This can be achieved by restricting the size of the symmetric key used such that an exhaustive search can be finished in feasible time. Every puzzle contains an id and a session key, both chosen randomly, as well as a static string, known to all participants.
The party initiating the key exchange, called X, generates n such puzzles and sends all of them to the receiver Y. Y picks one puzzle at random and decrypts it by trying all possible keys. Because of the static string inside the puzzle, Y knows for sure when the correct key has been tried. Y then extracts the session key and sends the corresponding id back to X. Subsequently, both parties can use the session key referenced by the id for encryption. An eavesdropper Z, monitoring all puzzles, cannot directly determine which of them is containing the returned id and therefore does not know the session key the two parties agreed on - the only possibility for Z is to attack **all** puzzles, squaring the effort spent by X and Y.
If, for example, one puzzle can be broken by 2^{32} computations, and 2^{32} different puzzles are used, X must prepare, save and send 2^{32} puzzles to Y, who in turn must try 2^{32} different keys. Z must crack all 2^{32} puzzles, each with effort 2^{32}, thus resulting in 2^{64} processing steps.
While this algorithm is very wasteful in regards of processing power, memory and communication capacity, such a protocol would be useful if a more-than-quadratic blowup could be achieved, but unfortunately, for all algorithms based on symmetric ciphers, this quadratic gap is the best that can be achieved, as shown in [33].
To further increase the effort an attacker has to spend a different approach has to be found. It turns out that hard mathematical problems exist that are suitable for such a purpose.
Therefore, in the next sections three important public key algorithms are introduced: D-H key exchange, based on prime fields, Elliptic Curve Cryptography (ECC), and RSA.

D-H Key-Exchange

Whitfield Diffie and Martin Hellman proposed a way to solve the problem for key-exchange based on finite fields when they published their paper *New Directions in Cryp-*

tography in the year 1976 [34]. The algorithm enables two entities to agree on a shared secret which never has to be transmitted between them. The security of their original algorithm is based on the hardness of the DLP, as shown in 2.2.

With the original D-H 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 both entities is to agree on the set of parameters $\{p, g\}$, where p is a large prime and g is a generator of the cyclic group ${Z_p}^*$. 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 $(1, q-2]$, 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 its counterpart's public key and raising it to the power of its own private key in the domain of ${Z_p}^*$:

$$\begin{aligned} k_B \equiv {y_A}^{x_B} \equiv (g^{x_A})^{x_B} \equiv g^{x_A * x_B} \pmod p \\ = \\ k_A \equiv {y_B}^{x_A} \equiv g^{(x_B)^{x_A}} \equiv g^{x_B * x_A} \pmod p \end{aligned}$$

An eavesdropper that intercepts the initially sent parameter 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 calculate the discrete logarithm of y_A or y_B to the base p, i.e. must solve the DLP, a hard problem as shown in section 2.2.

ECC

The D-H protocol can be based on different kinds of cyclic groups. Koblitz and Miller independently proposed the usage of cyclic groups based on ECs [35] [36]. See Section 2.5 for the theoretical introduction. This cyclic groups are receiving increased importance in cryptography because they allow the usage of shorter keys compared to D-H over prime fields or RSA, while providing the same level of secrecy.
To build a public key system, two users A and B initially agree on a set of public *domain parameters*. The most important parameters are [37]

- the order of the field, q
- the coefficients $a, b \in \mathcal{F}_p$, defining the EC
- the coordinates $(x_p, y_p \in \mathcal{F}_p)$ of the *base point* P,
- the order of P, denoted n

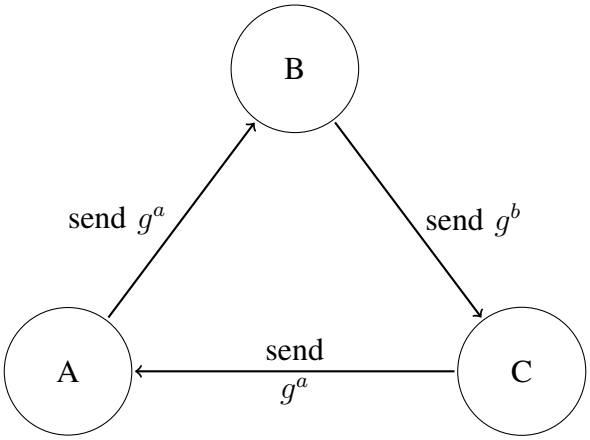


Figure 2.22: D-H Round 1

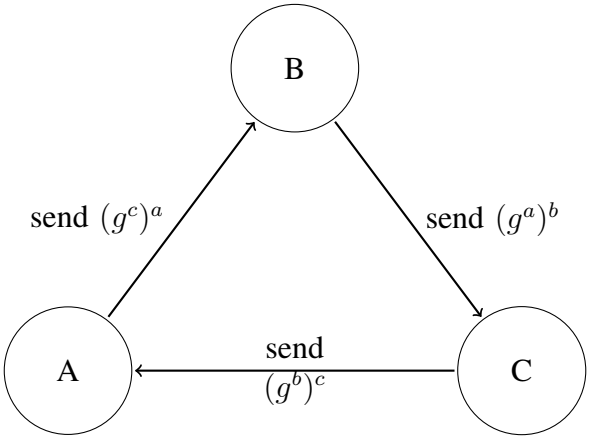


Figure 2.23: D-H Round 2

After agreeing on this set, A selects a randomly chosen integer k_A, calculates $P_A = k_A * P$ and sends this point to B, who in turn randomly chooses k_B and sends $P_B = k_B * P$ to A. Subsequently, both can calculate the point $k_A * k_B * P$, which can be used to derive a key.

Multi-party key negotiation

D-H and therefore also the ECDLP can be generalized to n parties, obtaining one key in common. The key negotiation procedure for 3 parties, using classical D-H, is shown in Figures 2.22 and 2.23, where it is assumed that all 3 parties A, B and C already agreed upon (p, g). After finishing the second communication round, every party raises the

last received value to its own private key and thus derives the shared secret.

$$((g^b)^c)^a = ((g^c)^a)^b = ((g^a)^b)^c = g^{a*b*c} \tag{2.8}$$

This algorithm can be generalized to n parties, using $(n-1)$ communication rounds. Obviously, this algorithm may not be practicable for a large n. Additionally, a complete new run must be executed whenever a new node joins.
Additionally, this key agreement mechanism (as well as all others presented so far) uses no authentication and is therefore vulnerable to active attacks.

RSA

RSA, published in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [38] and formalized in [39], relies on the hardness of finding the prime factors of a big composite number. In contrast to D-H, RSA is no to key agreement algorithm but can instead be used to encrypt *and* sign messages.
For key generation, two large primes p, q, which should be of about same size, are chosen randomly, with $N = pq$. Additionally, a public exponent e and a private exponent d are chosen s.t. they are multiplicative inverses to each other in $\pmod{\varphi(N)}$:

$$e * d \equiv 1 \pmod{\varphi(N)} \tag{2.9}$$

$\varphi(N)$, Euler's totient function, counts the number of integers in the interval $[1, N]$ which are relatively prime to N. For a prime p, $\varphi(p) = (p-1)$, therefore for the product $p * q$ of two different primes, $\varphi(p * q) = (p-1) * (q-1)$. Additionally, Euler's theorem is used:

$$a^{\varphi(N)} \equiv 1 \pmod{N} \tag{2.10}$$

The public key consists of the pair

$$(N, e)$$

and the private key of the pair

$$(N, d)$$

In practice, for the public exponent e the numbers 3, 5, 17, 257 or 65537 are suggested [40], a suitable d satisfying 2.9 can then be found by using the extended Euclidean algorithm.

Encrypting of messages

To encrypt, the message M must be converted to an integer. Then, the sender uses the recipients public key and raises M to the power of $e \pmod N$:

$$C \equiv M^e \pmod N$$

To decrypt, the receiver uses his own private key to raise C to the power of $d \pmod N$:

$$M' \equiv C^d \equiv M^{d*e} \bmod N \tag{2.11}$$

From the way e and d have been chosen in 2.9 it follows that

$$e * d = k * \varphi(N) + 1, k \in \mathbb{Z} \tag{2.12}$$

Inserting 2.12 in equation 2.11 yields:

$$M' \equiv M^{k*\varphi(N)+1} \equiv M * M^{k*\varphi(N)} \pmod N \tag{2.13}$$

By using Euler's theorem 2.10, expression 2.13 shows that $M = M'$, i.e. decryption yields the correct value because

$$M' \equiv M * M^{k*\varphi(N)} \equiv M * (M^{\varphi(N)})^k \equiv M * 1^k \equiv M \pmod N$$

Digital Signatures

Digital signatures are, as its symmetric-key MACs counterparts, used to provide integrity. Most digital signature schemes are based on cryptographically secure hash functions, so the same requirements as listed in 2.3 must also hold here.

Nevertheless, due to the use of asymmetric keys, an important semantic difference between MACs and digital signatures emerges: for a MAC a key is shared by at least 2 entities. A digital signature, in contrast, is generated by utilizing the private key of an entity and is not thought to be shared with other entities. Therefore, digital signatures can also provide non-repudiation: this property allows to convince an unbiased "judge" that a message, signed by the sender, was indeed sent by this sender, i.e. the message was not forged by a third party. This is an important difference to MACs, where such an assessment is not possible.

RSA

By 'reversing' the encryption process, the RSA algorithm can also be used to generate signatures of a message. This is typically achieved by generating a hash value of the message and *encrypting* that hash with the *private* key. The signature is then attached to the message. Afterwards, every entity can verify the integrity by *decrypting* the signature with the *public* key of the sender, calculating of the hash of the message and comparing it to the decrypted hash.

Elliptic Curve Digital Signature Algorithm (ECDSA)

Based on the idea of Scott Vanstone, this signature algorithm is the ECC variant of the Digital Signature Algorithm (DSA), standardized by NIST [41]. Analog to the ECDLP, as shown in section 2.2, the domain parameters are public knowledge. The signer of the message chooses a random integer k and computes the coordinates (x_Q, y_Q) of a new point Q by multiplying $k * P = Q$. Afterwards, x_Q is converted to an integer, obtaining $r = x'_Q \pmod n$. Finally, by using his private key d, s is calculated:

$$s \equiv k^{-1}(h(m) + dr) \pmod n \tag{2.14}$$

This results in the tuple (r, s), i.e. the signature of m.

The receiver of the message can use the domain parameters and signature to verify the integrity by calculating:

$$\begin{aligned} x' &\equiv s^{-1}(h(m)P + rQ) \pmod n \\ x' &\equiv s^{-1}(h(m)P + rkP) \pmod n \\ x' &\equiv P\underbrace{(s^{-1}(h(m) + rk))}_{k'} \pmod n \end{aligned}$$

x' is converted to an integer an reduced $\pmod{}$ to obtain r'. If $r = r'$, the signature is accepted, proofing the authenticity of the message.

Key lenghts

[42]

TODO: BIRTHDAY PARADOXON

CHAPTER 3

Availability

3.1 Introduction

Availability measures the delivery of correct service as a fraction of the time that the system is ready to provide a service.
The services provided by the system through its service interface to the user(s) is described by the functional specification. Whenever the provided service deviates from correct service a system failure, called *outage* or *downtime*[1] occurs. A failure is defined to be caused originally by a fault inside the system, which may be dormant. Under special conditions, the fault *may* become apparent and lead to an error. Faults can be categorized into random and systematic faults: random faults are unpredictable and concern hardware: for example, because of aging, a memory cell can be damaged, but the fault may be hidden because the cell is not used. Software and design faults belong to systematic faults. Similar to the damaged memory cell, a software bug may only trigger an error under special inputs. In both cases, the error then causes a deviation from the required operation of the system or a subsystem. Finally, the error can cause a failure if the system fails to provide the correct service:

$$fault \rightarrow error \rightarrow failure$$

Availability is measured subjectively from the system user's point of view. While an optimal system may have availability of 1, this value is only of theoretical value because random faults can not be ruled out. Therefore, system failures are inevitable, and availability is often given in *x-9s*[2] notation, denoting the number of nines in the time fraction

[1] in contrast, *uptime* does not guarantee correct service - for example, a server can be 'up' but unreachable

[2] read 'one-nine', 'two-nines', ...

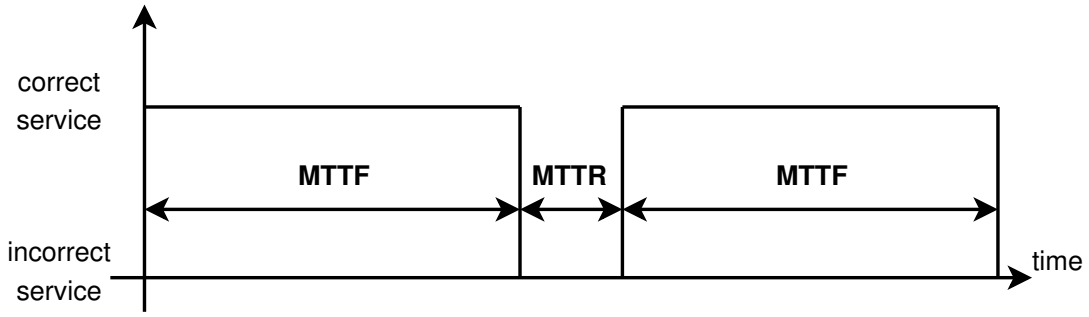


Figure 3.1: Relationship between MTTF and MTTR

the system delivers correct service, as shown in table 3.1. The availability may be asserted by the provider of the system to the customer by an Service Level Agreement (SLA) - if the SLA is violated, the provider may be fined.

x	$P_{availability}$	failure duration per year
1	90,0000	36 days
2	99,0000	3,5 days
3	99,9000	9 hours
4	99,9900	1 hour
5	99,9990	5 minutes
6	99,9999	31 seconds

For a system with 6-9 availability, this would mean that the system provider assures that the system will be unavailable not more than about 31 seconds over a whole year.

Availability is important simply because unavailable systems cause costs. Because of the relation of availability to reliability and the maintainability of the system, as shown in Equation 3.1 and Figure 3.1, two options to improve availability exist: either increasing the Mean Time To Failure (MTTF) (i.e. increasing reliability) or decreasing Mean Time To Repair (MTTR).

$$P_{Availability} = \frac{t_{correct\ service}}{t_{total}} = \frac{MTTF}{MTTF + MTTR} \tag{3.1}$$

Reliability measures the probability that the service will work as expected until time t:

$$R(t) = e^{-\lambda(t-t_0)} \tag{3.2}$$

3.2 is known as the *exponential failure law*, λ denotes the failures per hour, its inverse $\frac{1}{\lambda}$ is called MTTF. Reliability can also be expressed in terms of *un*reliability, denoted

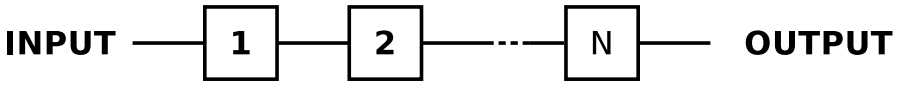


Figure 3.2: Model of a series system

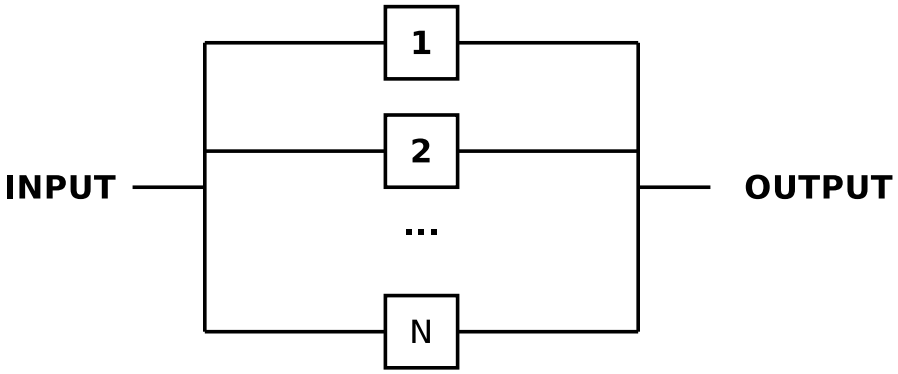


Figure 3.3: Model of a parallel system

$Q(t)$:

$$Q(t) = 1 - R(t) \tag{3.3}$$

Maintainability measures the time that is needed to repair a system, with μ denoting the repair rate and $\frac{1}{\mu}$ called MTTR.

Every system will most likely consist of different and distinct components, each with its own reliability and failure rates. The final system will possess an overall reliability, determining its availability. To model the resulting system, the components will be put together in a mixture of serial and parallel systems. For example, if all i components, ranging from 1 to N, are necessary to work correctly for the overall system to deliver correct service, this can be modeled as a series system, see Figure 3.2. This does not mean that the components are necessarily connected in a serial way, it just stresses that failure of one single components breaks the whole system. System's reliability then equals the product of all component's reliabilities:

$$R_{system}(t) = \prod_{i=1}^{N} R_i(t) \tag{3.4}$$

In contrast, a system containing redundant components, failure of one such component must not produce an outage of the overall system. This can be modeled as a parallel system 3.3. Here, the overall reliability can be calculated as given in 3.5.

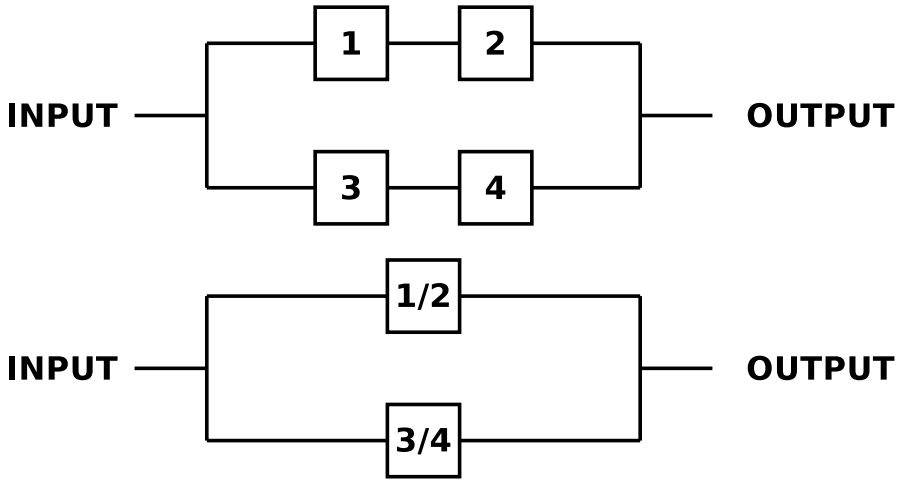


Figure 3.4: Condensing a mixed system

$$R(t)_{system} = 1 - \prod_{i=1}^{N} 1 - R_i(t) = 1 - \prod_{i=1}^{N} Q_i(t) \tag{3.5}$$

Mixed systems, containing paralell and series system, can be calculated by iteratively condensing serial- or parallel subsystems into single components, as shown in Figure 3.4.

High Availability (HA) characterizes a system that is designed in a way to avoid outages, or in case of a failure that can be repaired in shortest possible time. The level of the needed availability depends on the environment of the system and must be weighted against the additional costs, introduced by improving availability. Therefore, no hard definition for HA can be given - it is a design goal and depends on the context. Availability requirements will most certainly be higher for systems in safety-critical requirements. A safety-critical system must not fail, otherwise endangering human lives or causing substantial economic loss or environmental damage [43]. On the other side, systems that will produce only minor costs when not operating correctly will probably have to meet less stringent availability levels.
For example, our society heavily depends on electric power supply, an outage is nearly unacceptable. On the other side, it is impossible to guarantee 100% availability, so *if* a power outage occurs, the system must be fixed as soon as possible to restore correct service, otherwise risking human lives. On the contrary, a booking system will obviously result in financial losses for the company if not working, but no more serious consequences are to be assumed.

This work's focus lies on security-critical systems, increasing availability of KNX networks, but restricting its deployment to environments without safety-critical needs. Thus, safety criticality is neglected here because of its most stringent demands, as needed in avionics or weapons systems. The difference between safety-critical and security-critical systems can be given as follows [44]: safety means that software must not be harm the world (i.e. containment), while security means that the world must not harm software (i.e. protection).

3.2 Failure Avoidance

Different strategies exist to handle faults, thus trying to avoid that a fault propagates through the system and finally leads to a system failure. These strategies are introduced in the next section.

Fault Removal

Fault removal tries to identify faults by testing the system before its deployment.
Therefore, the system is exposed to test cases, which ideally would cover all possible internal states. Whenever a failure occurs, the erroneous state is identified, the underlying fault is removed and a new testing cycle begins.
A problem about this approach is the huge test space that for even simple systems would be required to iterate through. Therefore, the method suffers from the very fundamental dilemma that testing never can proof that a system is fault free.

Fault Avoidance

Fault avoidance aims at producing a system which is fault-free per design and is applied at the design stage of the system. Despite the methods for achieving fault avoidance can be applied in the domain of sofware and hardware, they can not guard against transient hardware faults occurring after the system is deployed.
Fault avoidance is based on two distinct processing steps: validation and verification. Validation is used to to show that the specification, which is the basis of system implementation, matches the real world within reasonable borders. This is necessary because every human built system uses an abstraction of the real world, thus simplifying the model.
In contrast, verification assures that the system indeed matches the specification. In other words, validation tries to answer if the correct system was built, while verification is concerned about to build the system correctly.

Formal methods can be be utilized for verification if the system model is available in a

formal language. The formal properties of the specification can then be checked automatically against a finite-state model of the system, a method called *model checking*. While automatic validation can proof that the resulting system is correct in regard to the specification, a big problem here is to obtain the formal properties out of an informal specification.

Obviously, both fault removal as well as fault avoidance cannot handle deliberately introduced faults, caused by an active adversary. Additionally, they cannot deal with hardware errors - therefore it is argued that fault tolerance is the only practical way to guarantee availability in a hostile environment (i.e. an environment where active attackers are assumed to exist and therefore DOS attacks cannot be ruled out), as well as to protect against random hardware errors.
Consequently, the proposed solution will be based on fault tolerance to achieve HA, which will be examined more detailed.

Fault tolerance

Fault tolerance tries to ensure correct service despite the occurrence of faults and is achieved through error detection and subsequent system recovery. To enable error detection, redundancy is added to a system and thereby the system's complexity is increased. This can be achieved both in the domain of hardware and software.
The basis of the design of fault tolerant systems is the definition of a *fault hypothesis*, stating which faults must be tolerable by the system, thus dividing the fault state into normal faults and rare faults (i.e. faults not covered by the fault hypothesis). Normal faults can be corrected by the fault-tolerance mechanism.

Fail-silent fault tolerance

Fault tolerance can be achieved based on *fail-silent* components. A fail-silent component either works correctly, i.e. outputs correct values, or does not output any values at all [45]. By duplicating such a module and comparing the outputs, fault tolerance can be achieved by cutting off the faulty module from the system.
This method is used for example for Redundant Arry of Independent Disks (RAID) based date storage [3]. For RAID level 1, all data to be stored is duplicated to two independent disks. In case a hardware failure of one drive is detected, the data can be accessed from the second drive. This method does obviously not work if both drives seem operable, but one drive outputs bogus data, i.e. it is not acting fail-silent. To han-

[3] although the logic distributing the data chunks to different devices can be built in hardware or in software, i.e. the operating system, RAID always relies on redundant disks

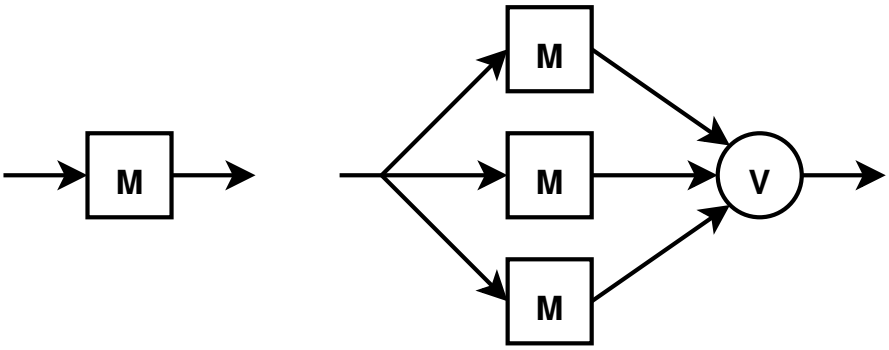


Figure 3.5: Non-redundant operation vs. TMR

dle this situation, higher redundancy levels are needed, as implemented for example in Tripple Modular Redundancy (TMR).

TMR

A TMR system, see Figure 3.5, is composed out of 3 modules or *black boxes*, all performing the same task, and one *majoriy organ* V, as proposed by Von Neumann [46]. The latter element is also called *voter* because out of its 3 inputs, it chooses the 'correct' output based on majority voting. As long as at least 2 black boxes do not fail, the system can provide correct service. If it is assumed that the voting element has perfect reliability = 1 and all black boxes M are independent from each other and have reliability R_M, the overall, time-invariant system's reliability is given in 3.6 [47] and plotted in graph 3.6

$$R_{system} = {R_M}^3 + 3{R_M}^2(1 - R_M) = 3{R_M}^2 - 2{R_M}^3 \tag{3.6}$$

It can be seen that if $R_M \leq 0.5$, overall reliability even worses, while for components reliabilities nearly being unit, very high system reliability can be achieved.

The concept of TMR can be generalized to n devices performing the same operation. In most cases n will be odd, so failure of at least $\frac{n-1}{2}$ modules can be tolerated.

3.3 Fault Tolerant Technologies

In the next sections, communication protocols based on ethernet, satisfying the needs of industrial communication networks, are examined. Beside the need for high availability, industrial production lines are sensitive against network disruptions. While, depending on the application domain short interruptions in the range of milliseconds to seconds may be tolerable, longer outages will force emergency stops or even cause damage.

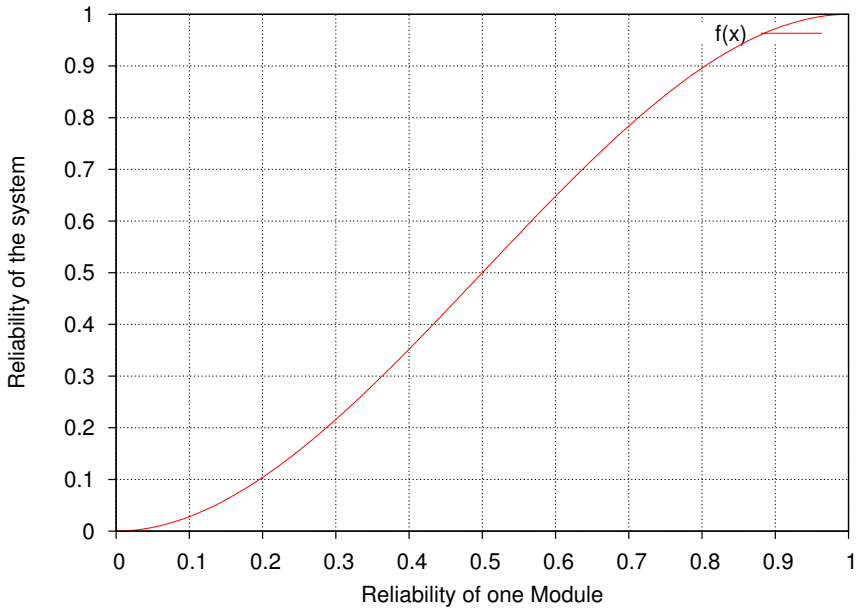


Figure 3.6: Reliability of resulting system

Network redundancy

Communication networks can be modeled as graphs. A graph, as shown in Figure 3.7, is an ordered pair $G = (V, E)$, with V being the set of vertices and E the set of edges, connecting the vertices. For a weighted graph, every edge is associated to a *cost*.
A *closed walk* or *cycle* exists if a path from one vertex to itself exists, resulting in multiple paths from one node to others. Such a network posses intrinsic redundancy, a fact that can be exploited to gain fault tolerant systems.
If the graph is undirected, every line determines a bidirectional communication link.
A spanning tree of graph G is a subgraph T, possessing all vertices V but only a subset of the edges $E' \subseteq E$ s.t. all vertices are reachable, but no loops exists. In the case of a weighted graph, the weight of the spanning tree is the sum of all existing edges.

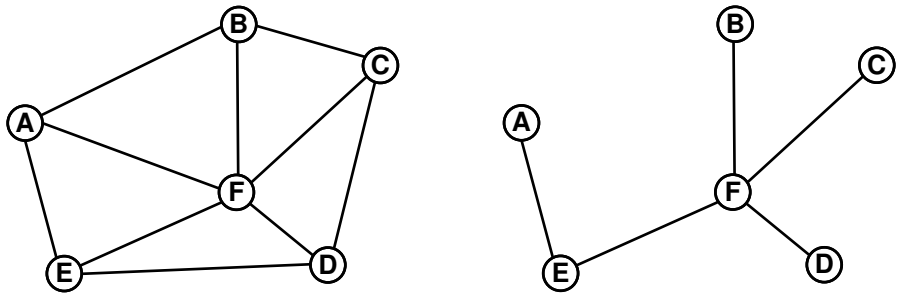


Figure 3.7: Undirected graph, spanning tree of the graph

Spanning Tree Protocol (STP)

The widely used Institute of Electrical and Electronics Engineers (IEEE) 802.3 ethernet standard was not designed with high availability in mind. Nevertheless, it already provided basic fault tolerance mechanisms based on the IEEE 802.1D STP.

In typical ethernet installations, depending on the network topology different nodes may be reachable through different paths, i.e. physical loops may exist. Switches, operating at Open System Intercommunication Model (OSI) layer 2, are responsible for loop detection and loop prevention and must logically disconnect such loops by blocking the corresponding ports to protect the network, as required by the ethernet specification, because multicast or broadcast traffic, generated by one of the connected devices, will be forwarded by all switches on all ports (except the incoming ports), thus flooding the network until no regular communication will be possible any more.

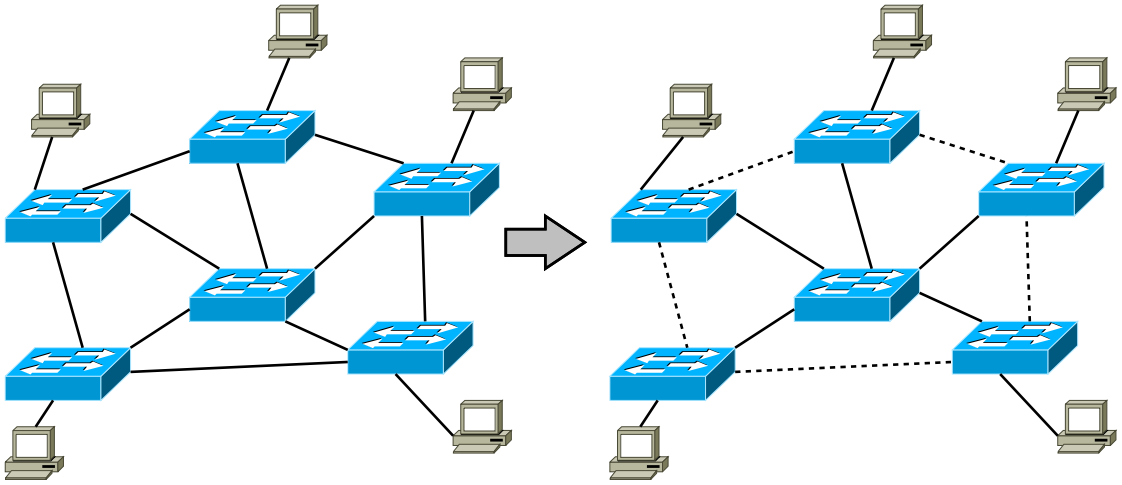


Figure 3.8: Protecting ethernet segments from loops with STP

This situation is shown in Figure 3.8: on the left side, the physical connections are shown, containing loops. STP removes the loops by setting certain interfaces to blocking mode, connected by dotted lines, as shown on the right side of Figure 3.8. The algorithm works as follows:

- The first step of the STP algorithm is to nominate one device as root bridge[4]. This can be done manually by the network administrator, or dynamically through the exchange of packages containing a priority. All switches will agree on lowest priority device as root bridge, additionally using the unique hardware MAC address if collisions occur. The root bridge sets all interfaces to forwarding mode.
- Afterwards, every switch determines the path with minimal costs for reaching the root bridge. The cost of every connection can be determined by the link speed or configured by the network administrator manually
- Finally, all devices except the root bridge set the ports belonging to the minimal-cost path to forward mode. All other ports are set to blocking mode, thus removing any loops.

If a connection is lost and an ethernet segment is not reachable any more, the STP, although not primarily designed for that tasks, can be used to handle that fault. Nodes affected by the link-change report that event to the root bridge, so that the network can

[4]the terms *switch* and *bridge* are used synonymously, with the major difference that switches can break network segments into multiple subsegments through Virtual Local Area Networks (VLANs)

be re-configured and a different path can be established, thus re-connecting the unreachable segment and providing fault tolerance.

A big drawback of this mechanism is the topology-dependent time needed for network re-organization. While in simulations delays of millisecond magnitude can be achieved [48], in practice link recovery can take up to one minute which is not acceptable for many industrial environments. An improvement was achieved with IEEE 802.1w Rapid Spanning Tree Protocol (RSTP) and a variety of different proprietary protocols, not compatible to each other, but still no sufficient recovery times were achieved [49].

IEC 62439

As it turned out that the mechanisms based on STP and its variations were not sufficient for many applications, the family of IEC 62439 standards was defined. IEC 62439 introduces a set of ethernet extensions, assuring high availability and enabling its deployment in industrial applications. Three extensions are introduced below.

Media Redundancy Protocol (MRP)

After IEC 62439-1 specifying basic definitions, the second draft introduced MRP which confines network outages to less than 500ms [50]. This is achieved by restricting the valid topologies to ring-only, so all switches are connected to the network by 2 interfaces. All switches beside the ring manager forward all traffic received, while the ring manager opens the loop logically by setting one of its interfaces to blocking mode. In this mode, all packages except management messages are discarded. The ring manager detects a failure by periodically sending test management messages in both directions. Additionally every node is able to detect a link failure on its local ports. It then sets the port to blocking and signals this link failure to the ring manager which opens his second port by setting it to forward mode and therefore reconnects the unreachable segment.
MRP does not rule out package loss in case of a failure, therefore it can be seen as improved RSTP for ring topologies.

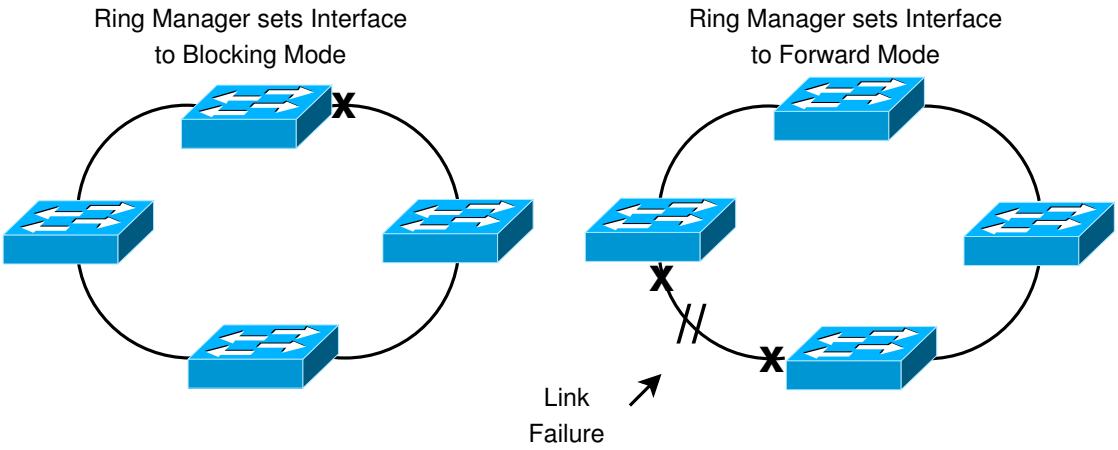


Figure 3.9: MRP fault tolerance

Parallel Redundancy Protocol (PRP)

In 2010, IEC 62439-3 defined a first version of PRP and High Availability Seamless Redundancy (HSR), both suitable for hard real-time systems [5] [51].
In 2012, IEC 62439-3 was revised, defining new versions of PRP and HSR. The basic ideas behind both protocols remained unchanged, so the latest versions are described below.

PRP can tolerate a link fault without package loss by utilizing two redundant and independent Local Area Networks (LANs). Nodes needing high availability, called Doubly Attached Node with PRP (DANP), are connected to both networks by two interfaces, sharing the same MAC and IP addresses. Additionally, standard ethernet devices called Single Attached Node (SAN) can be connected to only one network, enabling communication with devices on the same LAN only. An example of such a network is shown in Figure 3.10.
To support redundant connections for DANPs, a sub-layer on OSI-layer 2 (i.e. the link layer), called Link Redundancy Entity (LRE) is defined. Upper level data arriving at the LRE of a DANP is duplicated and equipped with 6 bytes of control information, containing a 2 byte sequence number, LAN number, length information and a static suffix identifying the Link Service Data Unit (LSDU) as PRP traffic. The sequence number is used for duplicate detection: every node uses one global counter Ctr_{global} for outgoing messages, not discriminating the receiver. For every package sent, the sequence number is incremented. Additionally, every node maintains distinct counter values Ctr_{source} for every unicast source-address it receives messages from, as well as for multicast and

[5] hard real-time characterizes a system where missing of a dead line may result in a catastrophic event

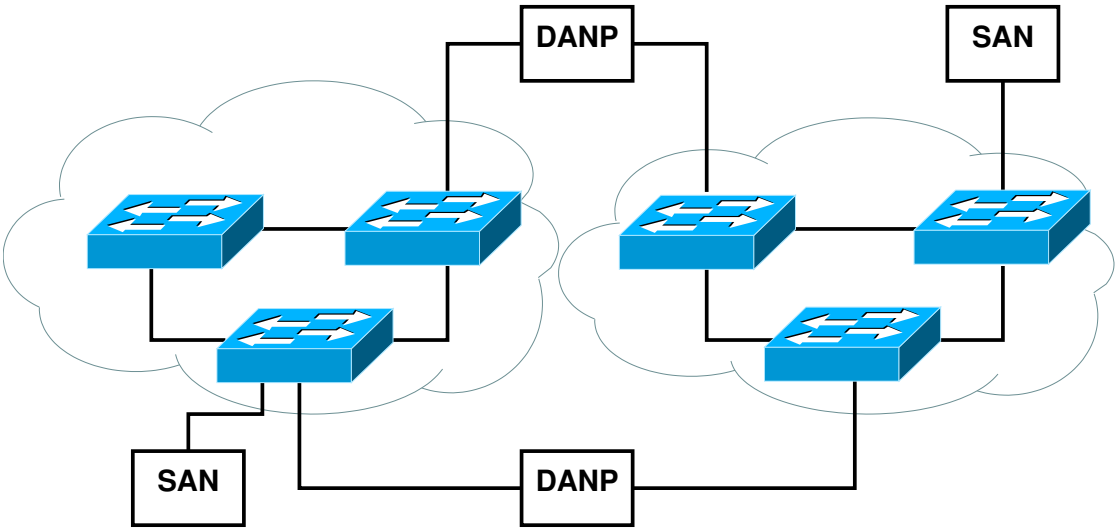


Figure 3.10: PRP network

broadcast messages.
On the receiving side the PRP traffic is detected because of the suffix, duplicates can be discarded by utilizing the sequence number [52]. The standard does not dictate how this must be achieved, it only demands that legitimate packages must not be discarded. Because of the short global outgoing sequence number, duplicates may not be detected as such - responsibility for final detection is delegated to upper layers.

HSR

HSR is based on ring-topology, but it also allows to connect HSR and PRP networks, different HSR rings or even to build a ring of rings. Every node is connected to the network by two interfaces, duplicating data from upper layers and sending one copy in clockwise, the other copy in counter-clockwise direction, thus eliminating the need for duplicated networks but wasting about half of the available bandwidth [53].

CHAPTER 4

KNX

4.1 Introduction

KNX [1] implements a specialized form of automated process control, dedicated to the needs of HBA. KNX 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) / OSI. Different kinds of physical backends are supported, allowing its use in different environments.
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 Points (DPTs), 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 therefore 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

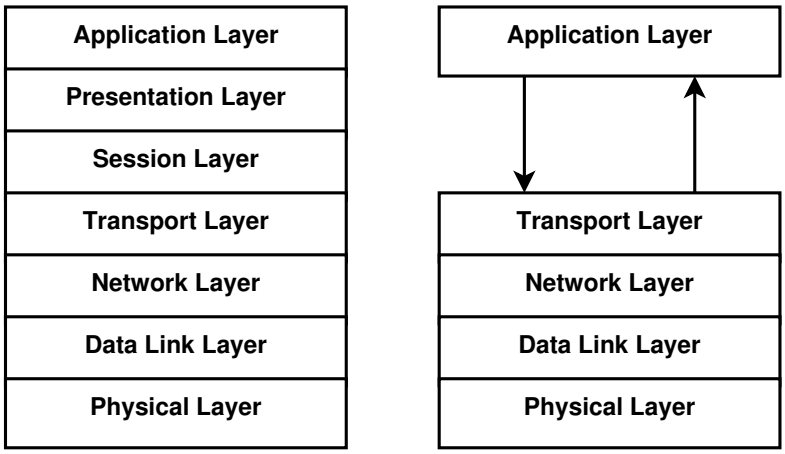


Figure 4.1: OSI Layer Model, compared to the KNX Modell

4.2 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 4.1. 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. 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 asyn-

[2] latin: *dive and rule*

chronously, with bidirectional, half-duplex communication and a datarate of 9600 bits per second (bps). TP-1 uses collision avoidance, and allows all topologies beside rings. 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/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. [54]

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

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

Control field

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

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

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

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

Figure 4.2: Control Field

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 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 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 (IA), 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 4.3 and extended frames, see Figure 4.5. While standard frames can bear up to 15 bytes of application data, extended frames allow the transmission of up to 254 bytes of data.

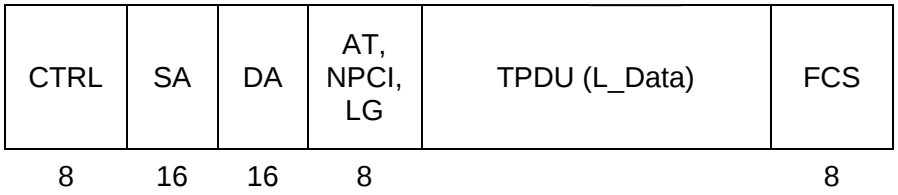


Figure 4.3: Standard Frame

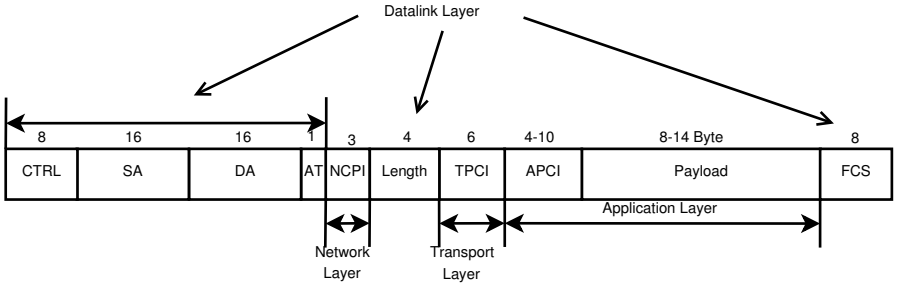


Figure 4.4: Standard Frame, in detail

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 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 4.6. 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, resulting in a maximum payload of 254 byte. After the length field, the payload and the check byte, as defined above, follow.

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

Figure 4.5: 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
1	r	r	r	0	0	0	0
1	r	r	r	0	1	x	x

Point-to-point Addressed L_Data_Extended Frame
Standard Group addressed L_Data_Extended Frame
LTE-HEE extended address type
All other codes are reserved for future use

Figure 4.6: Extended Frame CTRLE field

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 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. Therefore, such frames can only be used within one physical segment [55].

Acknowledge Frame

This frames are used to acknowledge the reception of a knx data frames for Group Addresss (GAs) or IAs 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: GAs or IAs, which type is used is determined by the 'address type' flag in the control field of the datagram(0 = IA). While the

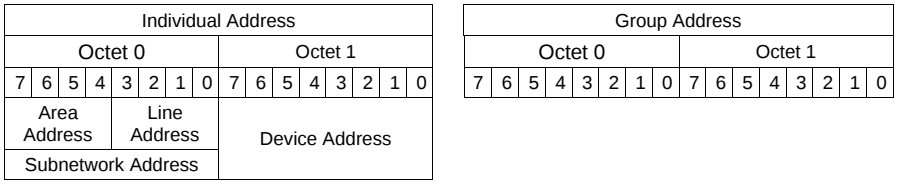


Figure 4.7: KNX individual and group addresses

source address always is an IAs and must be unique within the network, the destination address can be of type group or individual (see Figure 4.7 for the layout).

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) [56] [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 IA 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 TCP - is used, as shown in Figure 4.8.

All other modes are unreliable, i.e. unacknowledged, transport mechanisms and can be used to address a IA, a GA 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

[4] Originally, this was called Time To Live (TTL)

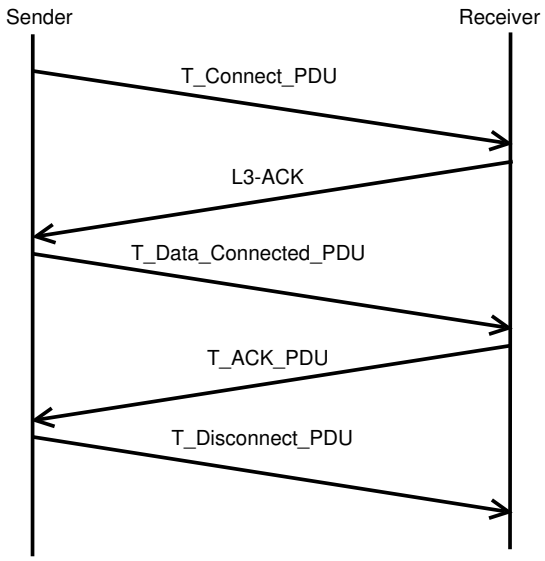


Figure 4.8: Handshake for connection-orientated communication

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.

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 useage of standardized DTs.

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.

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		
1								0	0	0	0	0	0		
1								0	0	0	0	0	1		
0								0	0	0	0	0	0		
0								0	1	SeqNo	SeqNo	SeqNo	SeqNo		
0								1	0	0	0	0	0	0	0
0								1	0	0	0	0	0	0	1
0								1	1	SeqNo	SeqNo	SeqNo	SeqNo	1	0
0								1	1	SeqNo	SeqNo	SeqNo	SeqNo	1	1

T_Data_Broadcast-PDU (destination_address = 0)
T_Data_Group-PDU (destination_address <> 0)
T_Data_Tag_Group-PDU
T_Data_Individual-PDU
T_Data_Connected-PDU

T_Connect-PDU
T_Disconnect-PDU
T_ACK-PDU

T_NAK-PDU

Figure 4.9: Flags used at the Transport Layer

4.3 Security in HBAS

HBAs emerged from automation systems, originally used for building control, summarized as HVAC. Central building management, leading to *intelligent buildings*, promises reduced maintenance costs, energy savings and improved user comfort [57], compensating the initially higher investment costs of such buildings. Following these arguments it would be natural to integrate additional building management functions like alarm systems, access control or communication systems, exploiting already existing infrastructure like cabling and thus benefiting from synergy effects.
This trend was contradicted by the fact that 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 [58]. Secondly, the possible threats by misusing HVAC applications were considered negligible. Additionally, the devices used in such networks were characterized by very limited processing power - thus, the comprehensive use of cryptographic measurements would have put remarkable computing loads onto these devices and was therefore considered impracticable.
Today the necessary processing power is available even on embedded systems, meanwhile systems integration continued until a point where security concerns could no longer be neglected. Therefore, the needs of HBA regarding security is introduced next.

Communication networks for HBA systems are usually built upon a two-tier model.
FIXME: not finished yet.

4.4 KNX security concept

As shown in section 4.3, a paradigm shift towards towards increased security in HBA can be observed. Despite that fact, the basic KNX standard 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."* [58]

For KNX/IP, the physical containment arguments do not apply. To counter this, it is proposed to use firewalls and Virtual Private Networks (VPNs) 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 [59]: 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" [60] which specifies the KNX Secure-Application Layer (S-AL), providing authentication and encryption, and the Application Interface Layer (AIL), implementing access control, both 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 AES128 is used in CCM mode, encrypting the payload with CTR and providing integrity with CBC mode. The overhead introduced by the 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 therefore 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 its 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 therefore 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 es-

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

KNX IP Security

This work focuses on securing the KNX IP specification, which can be used as backbone for connecting distinct KNX installations [61]. 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 4).
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).

4.5 Summary

Examining the security measurements shows up parallels to the IPv4 world. As with KNX, no security measurements were foreseen in the original standard. The problem was fixed here in two ways: TLS was added as a sublayer between OSI layers 4 and 5, allowing application-transparent encryption and authentication, similar to KNX Application Note 158, as described above. The second solution was the introduction of Internet Protocol Security (IPSec), extending IPv4 and thus enabling authentication and encryption on IP level.

Despite the KNX extensions, no solution providing high availability exists. Therefore, this work proposes to combination of cryptographic measurements with redundancy mechanisms, allowing its deployment in more demanding environments.

CHAPTER 5

Concept

5.1 Basic Assumptions

The aim of this work is to propose a KNX prototype, applicable for environments with increased availability demands. The prototype should be designed in a transparent way, utilizing a "plug and play" functionality to build a secure KNX network. This means that 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 (called cleartext KNX line) and two distinct connections to a secured KNX network (called secure lines, 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 two secure lines will constitute a secured KNX area, bridging between areas with increased security demands, as shown in Figure 5.1.

The basic tasks of such a security gateway consist of:

- establishing keys with its communication partners within the secured KNX network (the security gateways)
- providing redundant communication lines, achieving improved availability by 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 both secure lines
- checking all messages which are received on the secured lines for integrity and authenticity, removing duplicates, unwrapping and delivering them to the unsecured area

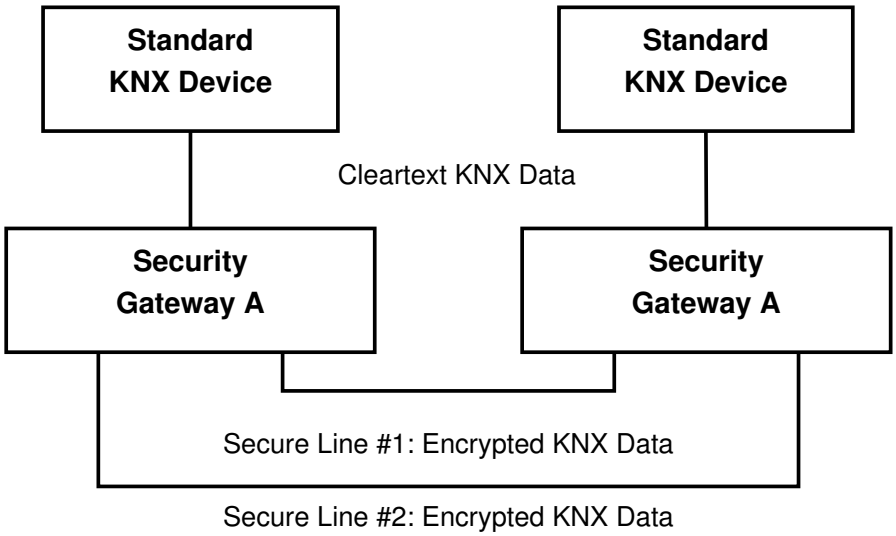


Figure 5.1: Secure Area

Security Related Architectural Overview

As stated in Chapter 4, 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 clear- to secure-KNX addresses (and vice versa) has to be defined. While in principle it would be possible to use the communication modes in a transparent way (i.e., 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 IA). 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 complicates 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 IA or a GA.

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 IA or GA 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 its 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 unsecure 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 2.22 and 2.23, a lot of messages have to be exchanged before an encryption can be done.

To encounter these problems, different keys must be used, thus achieving pairwise end-to-end encryption between all devices. Figure 5.2 shows the logical connections within a KNX network using end-to-end encryption. 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.

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 GAs and IAs of unsecured KNX devices to IAs of responsible security gateways . Such a routing table can be built statically at setup time, with the obvious disadvantage that the exact topology of the 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 IAs or GAs of unsecured KNX devices and IAs of security gateways at the border between the secured and the unsecured KNX network, as well as all keys used for the particular security level the gateway belongs to. This table would be generated once, after the topology of the network has been fixed, and 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 known group addresses behind existing security gateways. A new group address, introduced by a newly installed device behind an already existing security gateway, will not

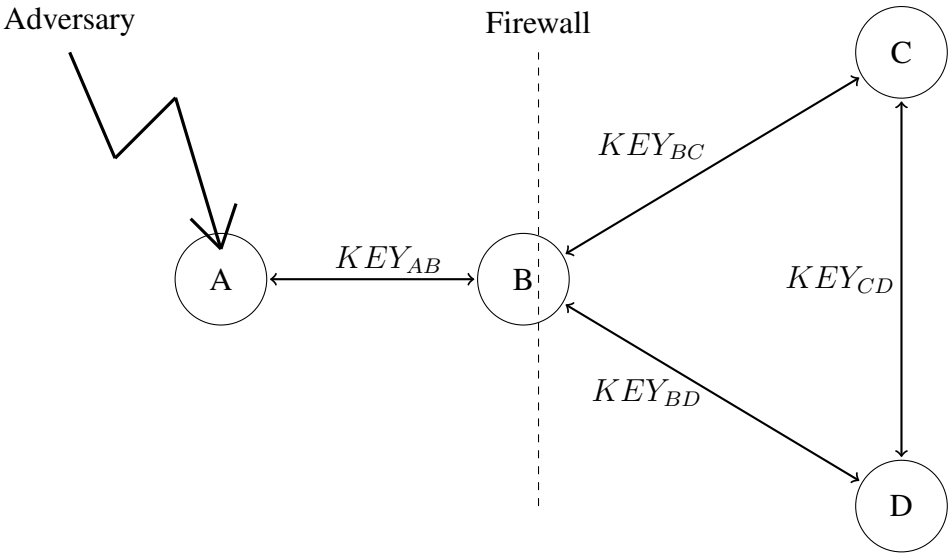


Figure 5.2: Firewall

be reachable, simply because the routing information is not available. Another disadvantage is that the deployment of new security gateways, connecting devices with new or already known GAs, is impossible as the IA 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 its unsecured lines for KNX devices, thus populating a list of reachable KNX devices. Whenever a device wants to send data to a group address, it has to process a lookup first to obtain the IAs of the responsible security gateways: the lookup must contain the wanted group address, as well as the sender's 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 its own public key, thus allowing pairwise end-to-end encryption. 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 introduced into the network.

A middle course is chosen: the reachable group address list is generated whenever a new security gateway is added to the network, handling discovery of these GAs as described above. This allows to deploy new security gateways with connected unsecured devices, thus achieving a compromise between flexibility and complexity.

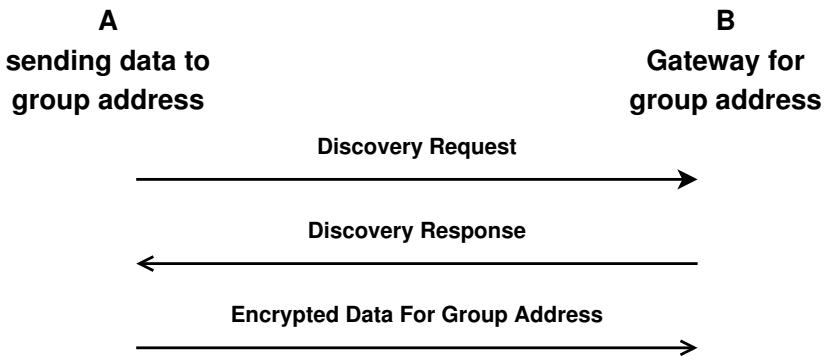


Figure 5.3: Communication schema

Sending data to a group address therefore follows the triad discovery request, discovery response and data transfer, shown in Figure 5.3. Broadcast messages are depicted as solid end of the arrow, the rest denoting unicast messages. To enable multiple devices to announce responsibility for a group address, the device wanting to transmit data to this group address must accept discovery responses following its request message for a short time window.

The discovery messages generated by security gateways should 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 further gets knowledge that this group address is responsible for switching a light (i.e. by visual observation), the attacker afterwards may be able to derive a personal profile just by detecting packages for this group address, although the payload of the datagrams to the responsible security gateway are encrypted. If the discovery messages are encrypted too the adversary doesn't know how many and what group addresses are behind a specific gateway, and it will be harder to derive personal profiles or to gather knowledge of the network topology.
Discovery request messages must be broadcast messages, readable by all security gateways. To limit the protocol overhead, a global network key is used here.

To provide authenticity, all datagrams passing the secure KNX network must contain a MAC to prevent modification of them.
Defense against replay attacks is achieved by counters. These must be strictly monotonically increasing and must not overflow. The counters can be compared to an initialization vector that prevents the mapping of same cleartext messages to same ciphertext messages under the same encryption key.

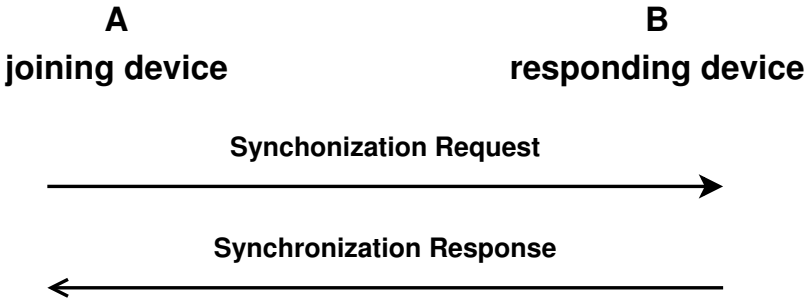


Figure 5.4: Synchronization service

Two different types of counters are used: one global counter Ctr_{global}, used for avoiding replay attacks against discovery messages. A second kind of counter is used for the actual data transfer. Beside avoiding replay attacks, this counter is necessary to detect and delete duplicates, caused by the redundant network.
Usage of the global counter Ctr_{global} raises another question, namely how a new device gets knowledge of the actual value of Ctr_{global}. Therefore, a synchronization service must be defined, allowing a newly powered up device that wants to join the network to synchronize with the rest of the network, as shown in Figure 5.4. Replay attacks are mitigated by including the current time into MAC calculation. Low temporal resolution relaxes time-synchronization issues between different devices.

Key Management

While it would be possible to use a centralized concept, no trusted on-line party is used in this work. 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. Otherwise, the network would suffer from a single point of failure in case no such fall-back mechanism is applied, an assumption that would clearly disqualify the design as highly available.
The following keys are used:

- A long-term key known to all security gateways is used. As already stated, this key must be copied to every device at setup time. This pre-shared key k_{psk} is used for symmetric encryption and serves different purposes: first, it is used to authenticate synchronization messages. Secondly, this key k_{psk} is used to encrypt discovery requests and discovery responses.

- Asymmetric keys are used for end-to-end encryption of the actual data packages between 2 security gateways. Elliptic Curve Diffie-Hellman (ECDH) serves as key negotiation algorithm. To protect against man-in-the-middle attacks, authenticity of the D-H parameters must be assured.

- This is the task of the third kind of keys: another pre-shared key is used to authenticate the D-H parameters.

Redundancy Related Architectural Overview

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 both lines. If both lines are available, i.e. there is, for example, no shortcut, the security gateway responsible for forwarding the frame 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, called Ctr_{out}, and is sent along the message. This counter must be unambiguously referenced by the origin cleartext message. The receiving side must maintain a counter for incoming packets, called Ctr_{in}, which will be updated by the received counter value as soon as the first frame is received if the received value is higher than the saved one. Subsequent delivery of the duplicate can be detected because the containing counter value equals the saved counter value.

To identify duplicate frames, basically various possibilities exist:
Referencing both Ctr_{out} and Ctr_{in} by the GA of the origin cleartext message does only work if for every destination GA in the network, at most one sending client device exists. Assuming client device A and afterwards client device B want to send the first message to the same destination GA, the delivery of the message originating from device A will trigger an update of the corresponding counter Ctr_{in} at the responsible gateway, but device B will use its own counter for the outgoing message. Because device's B outgoing counter is less than the gateway's actual incoming counter, both frames will be discarded by the gateway. Therefore, this is no viable solution.

It shows that the easiest way to unambiguously identify duplicates is by referencing both Ctr_{out} and Ctr_{in} by the IA of the origin cleartext message. This solution works despite potential network failures on one or both secure lines, provided that each client device is identified by a unique IA. This is argued as follows:
For simplicity, assume that 3 security gateways A, B and C are connected to each other by two distinct, secured KNX lines, and each gateway is connected to an arbitrary number of client devices through their cleartext lines, each with a unique IA. Additionally, each client device is destination for an arbitrary number of GAs. If a security gateway receives a cleartext message, it will at first check the counter value Ctr_{out} for the IA and increment it. After that, the discovery phase takes place. This discovery request

can be answered by zero, one or two other security gateways. If at least one reply is received, the package is duplicated, encrypted, fitted into a unicast data frame together with Ctr_{out} and sent on both lines.
Every gateway answering the discovery request will be sent 2 duplicate messages, one on each secure line. Now, there are 3 possibilities:

- If both secure lines are available, one data frame will be handled first and the contained counter will be saved as actual Ctr_{in} for the IA of the inner frame. When handling the second frame, the contained counter will be equal the saved counter, and thus the frame can be discarded.

- If only one secure line is available, no duplicate will arrive, but the receiving gateway(s) will nevertheless update the received counter value for the IA

- If both lines are unavailable, the responsible gateways cannot update the corresponding value for Ctr_{in}. Nevertheless, the sending side will increment and update the outgoing counter Ctr_{out}. As soon as the responsible gateways are reachable again, new data frames will bear an even higher counter than saved on the receiving side, thus allowing data transfer to the GA again.

Operational Constraints

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

- Acknowledgement frames, used in point-to-point communication, as defined in KNX and introduced in Chapter 4, 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 their unsecured lines. Only after that, all addressed, unsecured clients would be able to acknowledge the received frames to their local security gateways, which must forward 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

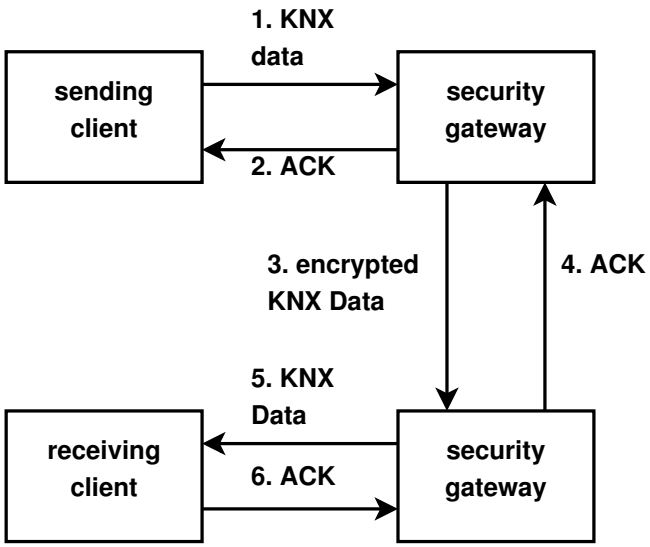


Figure 5.5: Acknowledgement of KNX data

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 time-out.

The only way to avoid a retransmission by a client is to immediately acknowledge a client frame by the security gateway that it is connected to, regardless of whether the destination device is reachable or not. The receiving security gateway will use a reliable transport mechanism to transfer the encrypted frame to all recipient gateways, which must acknowledge reception. Finally, the gateway on the receiving side will forward the contained frame to the client, who may generate an acknowledge frame, depending on the transport mode chosen by the sending device. This process is shown in Figure 5.5.

- Similar arguments avoid the processing of Poll-Data Frames. Here, event more stringent timing constraints are to be met, see chapter 4.

Apart from the data service, handling the actual data transfer of the KNX payloads, 2 additional services are necessary, following the assumptions above. These two services, handling synchronization and discovery, are defined as following and are summarized as management services. To distinguish the different frame formats, a 8 bit *secure header* field uniquely identifies every frame - see section 5.2 for details.

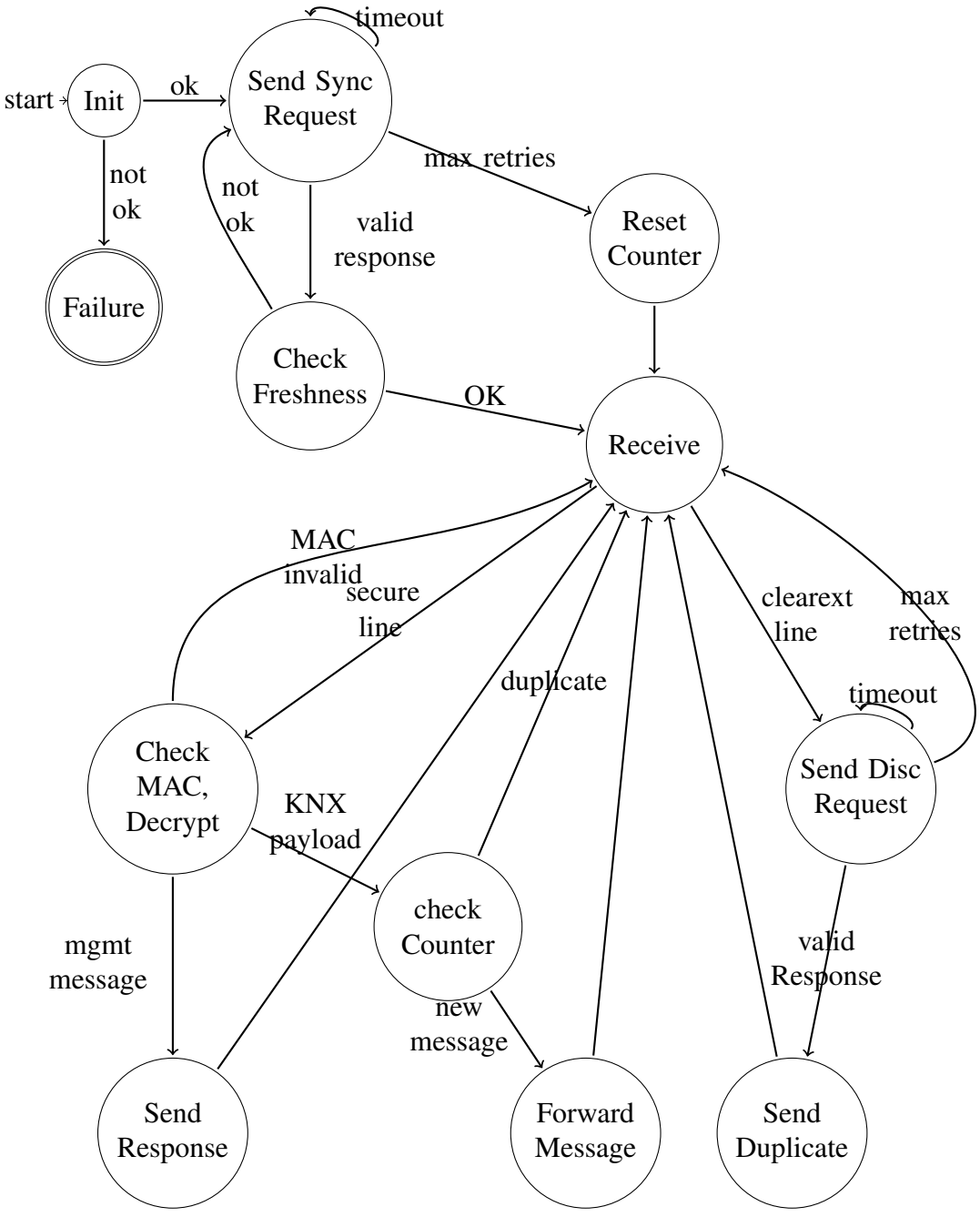


Figure 5.6: State machine of the program logic

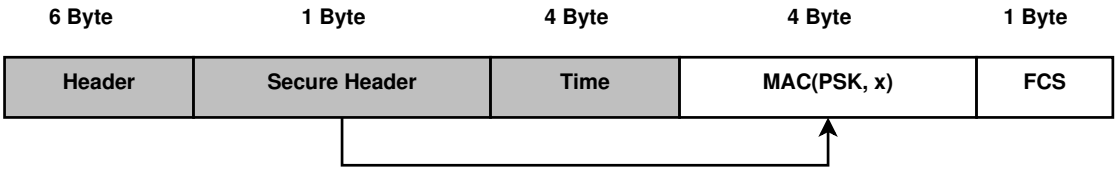


Figure 5.7: Synchronization request frame layout

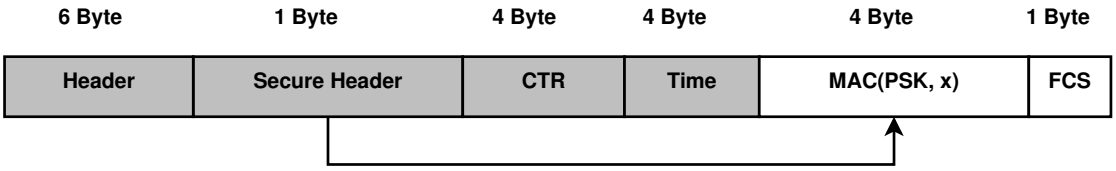


Figure 5.8: Synchronization response frame layout

5.2 Services

Synchronization service

As stated, a new gateway, joining the network, must get knowledge of the actual value of Ctr_{global}. This is achieved by sending a broadcast message on every secure line, serving as synchronization request. The frame contains the device's local time in seconds. The header flags in the secure header are set accordingly, identifying the frame as synchronization request, see Figure 5.7.

Every device receiving such a request checks the integrity of the message first by recalculating the MAC. Afterwards, freshness is checked by comparing the supplied time with its local time. If the timing information equals the device's own local time, the device sends a unicast synchronization response frame, containing its local time and the actual counter value. See Figure 5.8 for the layout of such a synchronization response. The accuracy of the time comparison is deliberately reduced by defining a window of allowed deviation. This allows a new device to join the network even if its local time and the local time of the answering device are not perfectly synchronized. If no synchronization response is received within 500ms, up to 2 retries are executed. After that, the device assumes that it is the first device in the network and resets the global counter Ctr_{global}.

The MAC is calculated over all frame fields except the trailing frame check fields and parts of the *Header* field.

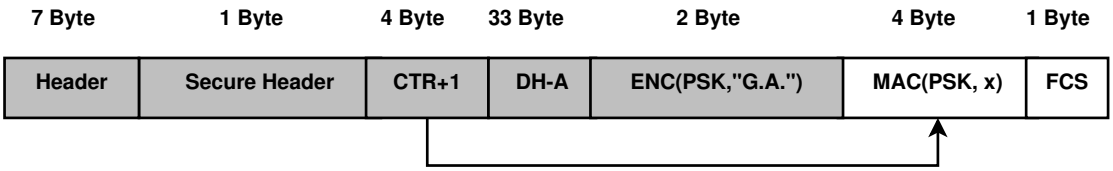


Figure 5.9: Discovery request frame layout

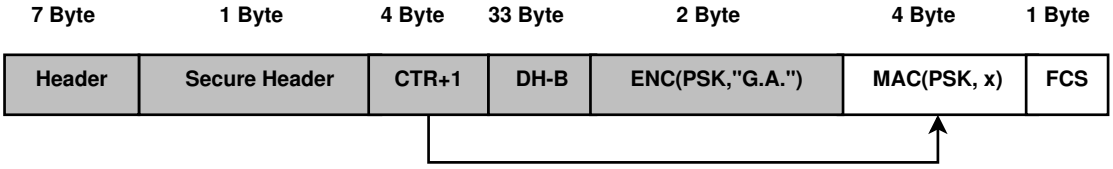


Figure 5.10: Discovery response frame layout

Discovery service

Whenever a gateway receives a message on its cleartext line, 2 distinct discovery broadcast messages are sent, one for each secured line. The frame format is shown in Figure 5.9. *DH-A* is the newly chosen ECDH - parameter of the requesting device. The PSK encrypted field contains the group address, CTR contains the incremented global counter Ctr_{global}. Every device in the network first checks the authenticity of the received frame by recalculating the MAC. If the value differs from the received one, the frame is discarded. Otherwise, the requested group address is obtained by decrypting the corresponding field, and every device recognizing the group address prepares a unicast response frame as shown in Figure 5.10, with *DH-B* its own newly chosen ECDH parameter and the incremented global counter in the *CTR* field. The requested group address is also sent, allowing the requester to identify the response message.
Integrity of the discovery messages is achieved by generating a MAC over all frame fields except the frame check field and parts of the *header* field.
The main logic from the discovery request receiver's point of view is shown in the state machine in 5.11

Data service

After receiving one or more discovery responses on one or both secure line, the device which wants to send the KNX payload (called requester) now knows the IAs which are responsible for delivering messages to the wanted GA. The requester can also derive pairwise shared secrets with all responsible gateways, based on ECDH. From this shared secret, a key is derived which is used to encrypt the origin KNX frame and inserted into

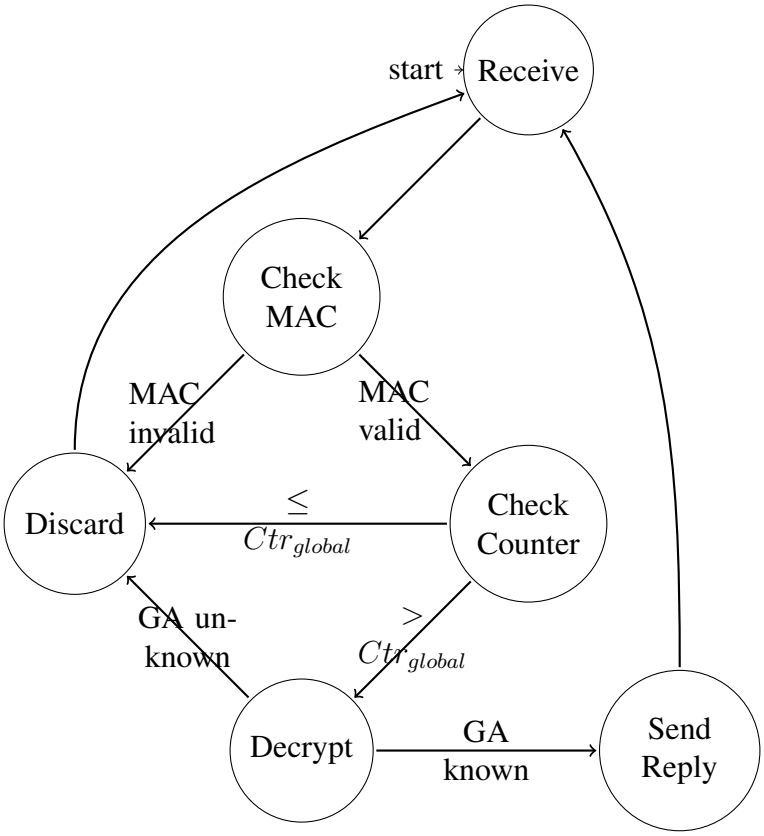


Figure 5.11: State machine for processing discovery requests

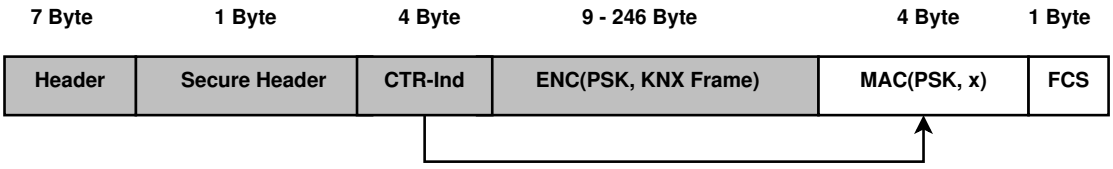


Figure 5.12: Data frame layout

the frame after the counter value. This counter $CTR - Ind$ is an individual counter, maintained by the gateway forwarding the KNX frame received over the cleartext line. Authenticity is guaranteed by calculating a MAC over all frame fields except the frame check field. The key used for the MAC is also derived from the shared secret.

Figure 5.12 shows the layout of such a data frame.

Data Structures

The used data structures, referenced above, are introduced in more detail below.

Secure header

Every frame sent by a security gateway contains a 8 bit header, uniquely determining the type of the frame. Implicitly, this information also determines the exact type of authentication or authenticated encryption mode used.

bytevalue	frame type	encryption	authentication
0000 0000	invalid	x	x
0000 0001	invalid	x	x
0000 0010	synchronization request	no	yes
0000 0011	synchronization response	no	yes
0000 0100	discovery request	yes	yes
0000 0101	discovery response	yes	yes
0000 1000	data service	yes	yes
0000 1001	reserved	x	x
...	...	...	...
1111 1111	reserved	x	x

Values 0x09 - 0xFF are not used - frames containing such values should be discarded.

Global counter

The global counter Ctr_{global}, obtained by the synchronization service and used in the discovery messages, is a 4 byte integer, allowing $2^{32} \approx 4,3$ billion discovery request or response messages to be sent before overflowing. This amount is assumed sufficiently high, argued as follows: each discovery messages consists of a frame containing 53 bytes, sent with 9600 bps, resulting in about 44 milliseconds transfer duration. Therefore, the absolute lower bound of the duration after that Ctr_{global} overflows, assumed the KNX network is occupied by discovery messages only, is about 2 years, a very conservative estimate.

Individual counter

Every individual counter is a 4 byte integer, responsible for duplicate detection. Two distinct types of individual counters are used: Ctr_{out}, referenced by an IA, and Ctr_{in}, also referenced by an IA. Figure 5.13 shows the state machine describing every security gateway's behavior from a high-level perspective.
Whenever a new KNX frame is received on the cleartext line and the gateway knows

which gateway(s) are responsible for the contained GA, the sending gateway determines the outgoing counter value $Ctr_{out}[IA]$. If an outgoing counter value for this IA exists this means that the device already sent at least one data frame to a GA, otherwise this is the first frame. For the first case, the counter value $Ctr_{out}[IA]$ is incremented and saved, otherwise it is set to 1. Afterwards, it is used as value for Ctr_{ind}. After that, the cleartext frame is encrypted and inserted into a new unicast data frame, one for each secure line.

Upon reception (the green transition in 5.13), at first the validity of the MAC is checked - if valid, the receiver checks its incoming individual counter value, referenced by the IA of the inner frame. If the received counter value Ctr_{ind} is greater than $Ctr_{in}[IA]$, Ctr_{ind} is saved as $(Ctr_{in}[IA]$ and the contained frame (i.e. the KNX data frame) is forwarded. The second frame, which will be handled afterwards will bear the counter value $Ctr_{ind} = Ctr_{in}[IA]$ and will be discarded, thus eliminating the duplicate.

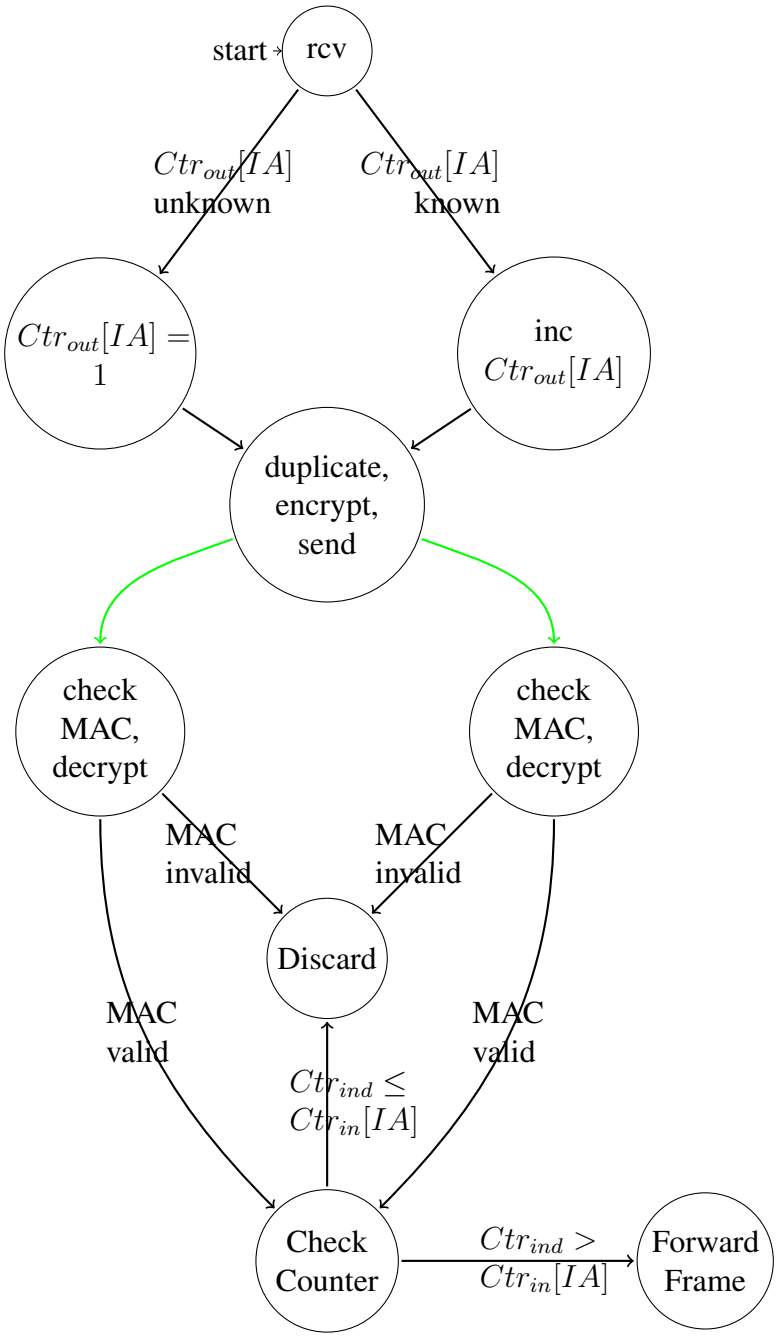


Figure 5.13: State machine of the duplicate detection logic

CHAPTER 6

Evaluation

To be able to evaluate the proposed solution in a real-life scenario, a prototype based on the concept presented was built. This chapter introduces the implementation from a high level view, followed by the evaluation of the result. Afterwards, the results are discussed and improvements are suggested.

6.1 Implementation

Bus interface

The necessary software, running on each platform, is written in the programming language "C". To interface with KNX, an Application Programming Interface (API) named European Installation Bus Daemon (EIBD) is used, providing functions to send KNX frames to and receive frames from the bus. EIBD offers synchronous as well as asynchronous calls for sending and receiving frames. While the first kind of call will block until the operation finished, the second kind of call will return immediately, the status of such calls can be checked by another special EIBD call. Because it is not possible to use callbacks with asynchronous calls, the implementation uses the synchronous functions.

Reading frames from the bus

EIBD provides a function to monitor the bus, this way all frames are available to the application.

```
1 EIBConnection *fd;
2 fd = EIBSocketURL(socketPath);
3 EIBOpenVBusmonitor(fd);
```

```
4 len = EIBGetBusmonitorPacket (fd, BUFSIZE, buffer);
```

Listing 6.1: Reading KNX frames

Writing frames to the bus

To write frames to the bus EIBD offers distinct functions for addressing individual devices or all devices in the segment. A connection must be first by calling the corresponding function, afterwards the data can be written to the bus:

```
1 // open an individual connection
2 EIBOpenT_Individual(fd, destination, 0);
3 // or a broadcast connection
4 EIBOpenT_Broadcast(fd, 0);
5 // finally, send the data
6 EIBSendAPDU(fd, len, (const uint8_t *)buf);
```

Listing 6.2: Writing to KNX frames

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 both lines per gateway, a different scheme is proposed. 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. 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.

Restrictions

EIBD can send extended frames, but can not receive frames with payload >= 56 byte. Additionally, it is not possible to set the source address of a written frame.

Concurrency

Every platform possess 3 distinct interfaces to the bus (2 interfaces form the redundant "secure" part of the network, one is connected to a standard KNX network), so care must be taken that no frames are missed or delayed due to a blocking call to EIBD. This can be achieved by splitting the main program into different processes or alternatively, threads, where for each critical task, a distinct part is responsible. While threads share

the same address space, facilitating communication with each other, processes rely on special system calls to share information. Additionally, thread-creation and switching between threads consumes less computing resources. Therefore, it was decided to choose the multi-threaded approach. Consequently, at least 3 different threads - one for each communication interface - are needed. Nevertheless, because every thread must be able to write to *or* receive frames from the bus at unpredictable moments, 7 different threads are used: the main thread only handles argument processing and creates the other threads. For the remaining threads, two pairs are responsible for the interfaces to the secured network, while the remaining thread pair interfaces to the unsecured part of the network. This setup is shown in Figure 6.1. This design allows to assign one timing-

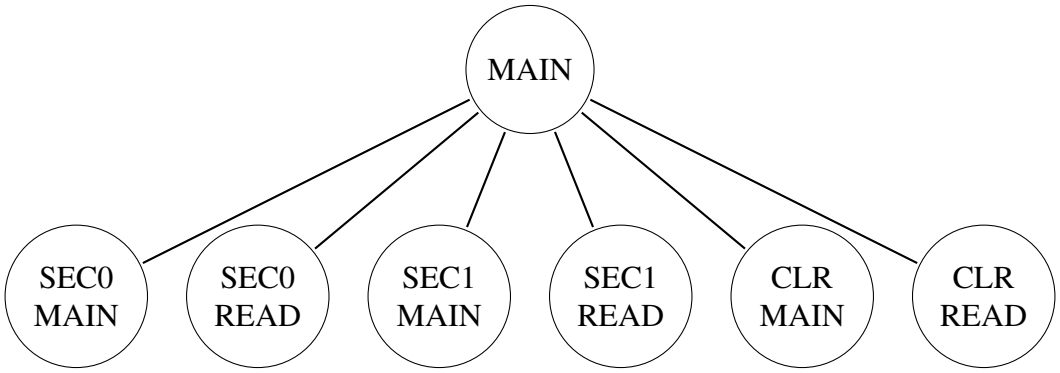


Figure 6.1: Threads used in the implementation

critical task to every thread: every *READ* thread immediately opens its corresponding bus interface in *monitor* mode, allowing it to read all the bus traffic on the corresponding line. After that, it enters a loop, performing a blocking read call to EIBD on every iteration. This will set the whole thread to *blocked*, and the operating system will set another thread or process as runnable. Whenever data is available to the blocked thread, the operating system will buffer the received data and eventually set the thread runnable again. Thus, no frames will be missed.

The main-threads of the secure lines (i.e. *SEC0 MAIN* and *SEC1 MAIN*) will start the synchronization sequence, as described in Section 5.2, and send out the synchronization messages. After that, the main threads will enter the *READY* state and listen for discovery messages.

Whenever the *CLR READ* thread receives new data, the frame will be forwarded to booth *SEC MAIN* threads together with the counter for the unique source address, contained in the frame. Booth *SEC MAIN* threads will send discovery requests, independently from each other, each containing newly chosen ECDH parameters. Responsible gateways (i.e. gateways which are connected to the wanted destination address through

their cleartext lines) will compute a new ECDH parameter, derive distinct shared secrets for booth lines, save them and send the parameters contained in the discovery responses back to the requesting device. The requester than derives the same shared secrets, and sends the encrypted frame, contained in extended KNX frames, to the destination gateways, which will check and decrypt them and forward booth decrypted frames and the corresponding global counter values to the *CLR MAIN* thread. Here, the duplicate will be discarded and one frame will be sent to its final destination.

A thread can be created in C with the function *pthread_create()*. The function expects 4 parameters:

- a variable of type *pthread_t*
- attributes for the new thread, or NULL for default values
- a function pointer, serving as entry-point for the new thread
- parameters for the function pointer, or NULL

```
1 if((pthread_create(&sec1MasterThread, NULL, (void *)secMasterStart, &
      threadEnvSec[0])) != 0)
2 {
3  printf("sec1Thread thread init failed, exit\n");
4  return -1;
5 }
```

Listing 6.3: Creating a new thread

If the call to *pthread_create()* succeeds, the new thread will start executing the function provided via the function pointer, and the main thread which called *pthread_create()* will continue. Which of the threads is running first and in which order the threads are assigned to the processor is unknown and determined by the operating system.

Thread-to-thread communication

To pass data from one thread to another, different ways are possible. While the easiest way would be to use global data structures, the implementation uses Portable Operating System Interface (POSIX)-pipes instead, mainly because with pipes, timeouts can be implemented easily. Such timeouts are used when waiting for synchronization replies, as well as for cleanup tasks.
How to create a new pipe is shown in Listing 6.4.

```
1 int pipefd[2];
2 if(pipe(pipefd) == -1)
```

```
{
  printf("pipe() failed, exit\n");
  exit(-1);
}
```

Listing 6.4: Creating a pipe

Every pipe possesses two file descriptors, providing a uni-directional data channel: data can be written to the pipe by performing a write() - operation to the *write*-end of the pipe (index 1), while the data can be read from the pipe by calling read() with the *read*-end of the pipe (index 0). Basically, the read() operation will block until new data can be read - in combination with the select() system call, a maximum time for the blocking operation can be set, as shown in Listing 6.5

```
fd_set set;
struct timeval timeout;
int selectRC;
...
FD_ZERO(&set);
FD_SET(fileDescriptor, &set);
timeout.tv_sec = 2;     // set timeout to 2 seconds
timeout.tv_usec = 0;
selectRC = select(FD_SETSIZE, set, NULL, NULL, timeout);
if(selectRC == 0)
{
   // timeout occured
}
else if(selectRC < 0)
{
  // error occured
}
else
{
  // data arrived - read it:
  read(pipefd[READEND], buffer, len);
}
```

Listing 6.5: Blocking read with timeout

In this example, FD_ZERO at first clears the set containing the file handles to be watched, FD_SET() then adds the file handle we are interested in (allowing to watch multiple file descriptors). The timeout is then set to two seconds (microseconds granularity is supported, nevertheless the timing precision is limited by the system clock granularity and kernel scheduling delays), and select() is called. If no data is received within 2 seconds, the if-branch is executed. If data is received in time, the else-branch is

executed instead, and the waiting data can be obtained by performing a standard read() call.

Race conditions

Software race conditions occur whenever an application, using shared resources, depends on the timing of its processes or threads. Because the exact time when a particular process or thread is scheduled by the processor is unknown in advance, such conditions must be avoided. An example of a program containing a race condition would be as following:

```
1 for ( int i = 0; i < 1000000; i++ )
2 {
3   x = x + 1;
4 }
5 // x = ?
```

Listing 6.6: Race condition

For a single-threaded program, this would produce the value $x = 1000000$ after the loop, while for a 2-threaded program, the resulting value will depend on the exact scheduling of the threads and very likely differ from the expected value $x = 2000000$. The reason is that the statement incrementing x is no atomic operation. Instead, the operation will consist of several statements on machine language level, forming a *critical region*:

1. load value from address into register
2. add 0x01 to the register
3. write value from register to address

If thread 1 gets interrupted after the second step and thread 2 afterwards finishes, thread 1 will overwrite the value just written by thread 2 and thus produce a false result.

Such a critical region must be protected such that only one thread can enter it, a requirement also called *mutex*, short for *mut*ual *ex*clusion. The implementation uses the POSIX *pthread mutex* to lock critical regions, as shown in listing 6.7:

```
1 if(pthread_mutex_init(&globalMutex, NULL) != 0)
2 {
3   printf("mutex init failed, exit");
4   return -1;
5 }
6 pthread_mutex_lock(&globalMutex);
7 /*
```

```
8    CRITICAL REGION
9 */
10 pthread_mutex_unlock(&globalMutex);
```

Listing 6.7: Locking a critical region

globalMutex is a global variable of type *pthread_mutex_t*, shared between all threads which are accessing the shared resource. After calling pthread_mutex_lock(), there are two possibilities: if the mutex is unlocked, the call will lock it and continue the program flow. If it is already locked, the call to pthread_mutex_lock() will block and the corresponding thread will be set to blocking. If the mutex is unlocked by the possessing thread, the blocked thread will be set active, lock for his part and continue execution.[1]

Cryptographic functions

To use the needed cryptographic functions, the open source library "OpenSSL" is utilized. The API supports a wide range of symmetric and asymmetric encryption, decryption, signing and key negotiation algorithms through two different kind of interfaces: a high level "EVP" interface, hiding much of the complexity, and a low level interface. If possible, the high level interface should be used, but for the key derivation function it turned out that the low level interface was easier to use.

Documentation and code snippets can be found on https://wiki.openssl.org, another important source of information are the man-pages (available under Debian after installing the package *libssl-doc*) and the header files.

It is noted that in the examples below no error checking of the return codes returned by the OpenSSL calls is executed. This checks are omitted here just for clarity, nevertheless it is imperative to always check them to identify error conditions.

Key derivation

To obtain a shared secret between two security gateways, ECDH over the curve "NID X9 62 prime256v1" is used. This is a NIST elliptic curve defined over the 256 bit prime $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, thus the resulting points will have coordinates of size 256 bits each. The uncompressed form of such a point would consist of 65 byte: 1 byte for a leading tag, 32 byte for the x and 32 byte for the y coordinate of the point. Because of the quadratic form of the curve (see Section 2.2 for a recap), it suffices to transmit only the x coordinate and one byte, determining the sign of the solution. The corresponding y coordinate can than be recovered.

[1] Of course, pthread_mutex_lock() must be executed in an atomic way, otherwise introducing a race condition on another level

Whenever a security gateway receives a new frame on its cleartext line, it generates a new ECDH parameter, i.e. it generates a new key pair, the public key is then sent to all other security gateways in the segment:

```
1  EC_KEY *pkey;
2  pkey = EC_KEY_new_by_curve_name(NID_X9_62_prime256v1)
3  EC_POINT *ecPoint = NULL;
4  size_t ecPoint_size;
5  const EC_GROUP *group;
6  EC_KEY_generate_key(pkey));
7  EC_KEY_set_conv_form(pkey, POINT_CONVERSION_COMPRESSED);
8  ecPoint = (EC_POINT *) EC_KEY_get0_public_key(pkey);
9  group = (EC_GROUP *)EC_KEY_get0_group(pkey);
10 ecPoint_size = EC_POINT_point2oct(group, ecPoint,
       EC_KEY_get_conv_form(pkey), buf, BUFSIZE, NULL);
11 if(!EC_POINT_is_on_curve(group, ecPoint, NULL))
12 {
13   printf("ERROR: point not on curve\n");
14   exit(-1);
15 }
```

Listing 6.8: Generating a ECDH parameter

The new key pair is generated in line 6, the public key is obtained and converted to a hexadecimal string in lines 8 and 10. Finally it is checked if the new point indeed is part of the curve.

The receiving side performs the same operations to obtain its own key pair which is sent back to the requester, and immediately derives the shared secret as shown in Listing 6.9.

```
1  size_t secret_len;
2  unsigned char *secret;
3  EC_POINT *peerEcPoint = NULL;
4  EC_KEY *peerEcKey = NULL;
5  const EC_GROUP *group = NULL;
6  peerEcKey = EC_KEY_new_by_curve_name(NID_X9_62_prime256v1);
7  group = EC_KEY_get0_group(peerEcKey);
8  peerEcPoint = EC_POINT_new(group);
9  EC_POINT_oct2point(group, peerEcPoint, peerPubKey, (size_t)33, NULL))
10 EC_KEY_set_public_key(peerEcKey, peerEcPoint))
11 secret = OPENSSL_malloc(32);
```

```
12 secret_len = ECDH_compute_key(secret, 32, EC_KEY_get0_public_key(
       peerEcKey), pkey, NULL);
```

Listing 6.9: Deriving a shared secret

Here, the public key received (the pointer *peerPubKey*) must be first converted from a hexadecimal string into point representation, and finally into an EC_KEY object (lines 9 and 10). The actual key derivation is shown in line 12.

This derived secret must be fed into a secure hashing function, for example SHA-256 because the shared secret may not be uniformly distributed. By appending different strings it is also possible to derive an encryption and a MAC key in one run.

MAC generation

For this task, a SHA-256 based HMAC is used, producing a 32 byte MAC. To reduce the overhead, only 4 bytes are used and contained in the messages. The usage of the high-level API is shown in Listing 6.10:

```
1  size_t req = 0;
2  EVP_MD_CTX* ctx = EVP_MD_CTX_create();
3  const EVP_MD* md = EVP_get_digestbyname("SHA256");
4  EVP_DigestInit_ex(ctx, md, NULL);
5  EVP_DigestSignInit(ctx, NULL, md, NULL, pkey);
6  EVP_DigestSignUpdate(ctx, msg, mlen);
7  EVP_DigestSignFinal(ctx, NULL, &req);
8  *sig = OPENSSL_malloc(req);
9  *slen = req;
10 EVP_DigestSignFinal(ctx, *sig, slen);
11 EVP_MD_CTX_destroy(ctx);
```

Listing 6.10: Generating a HMAC

Line 2 allocates and initializes a digest context, a digest structure for SHA-256 is obtained in line 3 and initialized in line 4. Line 5 sets up the signing context for the underlying digest function. The actual signing of the input message string pointed to by msg is done in line 6. Afterwards, the length of the signature is obtained by the first call to EVP_DigestSignFinal(), the signature is written to the newly allocated memory and the context memory is deallocated

Encryption and decryption

As cipher, AES-256 in CTR mode is utilized. This mode has the advantage that no fixed block size must be transmitted, but only the actual size of the cipher text. Encryption with the OpenSSL high level interface is shown in Listing 6.11. As with the HMAC, a context must be created and initialized.

```
1 EVP_CIPHER_CTX *aesCtx = NULL;
2 int cipherLen, len=0, i=0;
3 aesCtx = EVP_CIPHER_CTX_new();
4 EVP_EncryptInit_ex(aesCtx, EVP_aes_256_ctr(), NULL, key, ctrFull);
5 EVP_EncryptUpdate(aesCtx, &cipherBuf[0], &len, msg, msgLen);
6 cipherLen = len;
7 EVP_EncryptFinal_ex(aesCtx, cipherBuf + len, &cipherLen);
8 cipherLen += len;
9 EVP_CIPHER_CTX_free(aesCtx);
```

Listing 6.11: AES encryption

6.2 Evaluation

The proposed solution was evaluated by showing that it can resist against the attacks defined in Section 2.1, and is robust against temporarily or permanent failure of one out of two secure KNX line.

Failure of one secure communication line does not imply failure of the whole system: as long as the other secured line is still functional, synchronization, discovery and data services will be handled by the functional line alone. As soon as the broken line is available again, discovery and data service messages will be sent on booth lines.

For malicious attacks, it is assumed that an attacker only has polynomially bounded processing powers and is able to passively read frames from and inject arbitrary frames into both secure communication lines. In contrast, it is assumed that the cleartext KNX lines are out of reach of an attacker, as well as the hardware running the master daemons, especially the memory holding the long-term encryption and authentication keys. Attacking the hardware itself would allow the attacker to obtain these keys and thus present itself as legitimate gateway.

The overall aim is to show that the communication network is able to withstand maliciously introduced faults, as well as unintended faults happening randomly, resulting in a KNX network with improved availability. In the case of DOS attacks it is stated that the proposed solution can withstand such an attack against **one** of it's two secure communication lines because of the duplicate and independent sending of the corresponding messages. An attacker interrupting both lines will obviously cripple the data exchange in total. Such a DOS attack can be conducted for example by shortcutting the TP lines, or by permanently driving the bus line into the dominant level.

Synchronization phase

Passive attacks

Packages in the synchronization phase are not encrypted, allowing a passive adversary to learn the value of the global counter value Ctr_{global}. Nevertheless, this counter is only used to avoid deterministic encryption (see 2.3) and is of no use for the attacker. Additionally, the attacker is able to learn all header fields, in particular the device addresses of all active security gateways, but this is considered inevitable and also of little use for the attacker.

Active attacks

An active attacker can inject new synchronization request and response messages, but will fail to produce a correct MAC for the actual time stamp with probability $1 - \frac{1}{2^{32}}$ because the MAC equals a random 32 bit number for the given header and payload. Such a MAC forgery will be detected by all active security gateways, and the corresponding frame will be discarded.
Opening a window for tolerating clock deviations allows an active attacker to replay captured synchronization request and response packages within that time window. Nevertheless, this is considered uncritical: for synchronization request messages, the attacker can trigger a new synchronization response message by a legitimate security gateway, which will re-send the actual counter value to the source address of the replayed message. The corresponding device however has already finished synchronization phase and will just drop the message.
When replaying a synchronization response message within the valid time window there are two possibilities: if a joining device is waiting for a response message the replayed message will be handled as legitimate response, and the newly joined device concludes the synchronization phase. On the other hand, if no device is waiting for a synchronization response, the message will simply get dropped.

Discovery phase

Passive attacks

In this phase, a passive attacker is able to learn the global counter value Ctr_{global}, as well as the ECDH parameters exchanged by requester and responder. Booth is considered unproblematic: for the first case, the same arguments as given for the synchronization phase hold. To derive the key used by two parties in the subsequent data transmission from the D-H parameters the attacker would need to solve the ECDLP, as shown in Section 2.2.

Additionally, he can learn the *encrypted* value of the requested group address and also learns which gateway(s) are responsible for the encrypted gateway. It is argued that it is impossible for the attacker to learn the underlying cleartext group address because of the 256 bit AES encryption, assuming that the attacker does not know the long term encryption key. Nevertheless, the attacker is able to derive communication profiles, i.e. which devices behind which security gateways are communicating with each other, and therefore enables the attacker to derive the basic topology of the network.

Active attacks

Injecting new or replaying old messages is useless from an attackers point of view, argued as following: for newly generated, injected discovery request messages, the attacker must at first generate the encryption for the wanted group address. Lacking the encryption key, the attacker can produce the correct encryption for the 2 byte group address with probability $1 - 2^{16}$. Afterwards, the attacker additionally must guess the correct lowest 4 bytes of the 256 bit MAC for the header fields and payload (again provided that the attacker does not know the key used for MAC generation). Failure to forge the correct MAC will be detected by all receiving devices.
Similar arguments hold for discovery response messages.

Replaying discovery messages is considered pointless because of the freshness property provided by Ctr_{global}: such repeated frames will be detected by the security gateways because of the outdated value of Ctr_{global}, which will just drop the replayed frame.
Attacking alternating secure communication lines will also fail: for example, the attacker could at first shortcut one secure communication line s.t. the discovery message will not reach it's recipient(s). Receiving the discovery message on the other line and injecting the frame to the previously blocked line would result in a fresh counter value. Nevertheless, such frames will bear the wrong secure line number as part of the source address (for discovery request messages, with the destination field set to the broadcast address) or source- and destination address, and will thus be detected. Alternating the source- and/or destination-address will invalidate the MAC, forcing the attacker again to forge the MAC, an infeasible task as already stated.

Data transmission phase

Passive attacks

An eavesdropping attacker will be able to learn source- and destination addresses of the security gateways exchanging the frame, as well as the length of the inner frame and the individual counter Ctr_{ind}. Again, the meta data can be used to generate communication profiles, a fact considered inevitable. The individual counter is used as freshness

property and to detect the duplicate frames on the receiving side. Therefore, an attacker does not benefit from knowing this counter value.
Decrypting the contained inner frame is considered impossible base on the following facts: firstly, encryption is based on AES-256, therefore trying all possible keys is infeasible. Secondly, the attacker is unable to get knowledge of the key because of the key agreement protocol used in the discovery phase.

Active attacks

An attacker, trying to inject a new data frame, again must succeed in forging the correct MAC. Failure to do so will be detected by the receiving gateway. A replayed message will be be correctly verified and decrypted by the receiving device, but because of the outdated counter value Ctr_{ind} the message will be discarded.

Conclusion and outlook

It was shown that the proposed solution can withstand malicious attacks, as well as transient hardware failures of one of the secured lines. Therefore, the solution allows to connect standard KNX devices which are spatially divided in a secure manner, bridging over areas where malicious behavior cannot be ruled out. The proposed solution can be deployed in a 'plug-and-play' manner, as long as the constraints defined in Section 5.1 are regarded. Thus it is possible to add confidentiality, integrity and availability to a KNX network just where needed, coexisting with segments with low security. By using more than 2 secured lines, the solution can easily be extended to use n instead of 2 secure threads and communication lines, boosting availability to an even higher level.

The following improvements to the prototype solution are proposed:

- Using a cache for the mapping group address - security gateway / encryption key: this allows to reduce the bus load
- Obfuscating the size of the data service messages: this can be achieved by adding an additional length field and corresponding padding, chosen randomly. This makes it harder to derive communication profiles
- Attacks can be detected if frames with forged MAC are received, allowing to send an alert to an operator and adding the corresponding device address to a blacklist.

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:

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

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

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

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

```
# 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 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 EIBD API and therefore is needed to be linked to the master daemon.
Additionally, the development files for interfacing with the openSSL API must be installed by executing the following command:

```
# aptitude install libssl-dev
```

A.3 Revision control

The source of the master daemon is managed by GIT. GIT is a decentralized furevision-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.

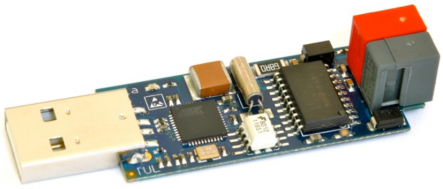
```
# git clone git@github.com:hglanzer/knxSec.git
```

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/ttyACMx*, 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
```

Figure A.1: Busware KNX-USB coupler



```
4 ... usb 1-1.2: Product: TPUART
5 ... usb 1-1.2: Manufacturer: busware.de
6 ... usb 1-1.2: SerialNumber: 7543034373135130C140
7 ... cdc_acm 1-1.2:1.0: ttyACM0: USB ACM device
```

If no such line like 7 appears, the correct firmware is available as file *firmware/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.

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

```
1 udevadm info --name=/dev/ttyACM0 --attribute-walk
```

A.6 Test setup

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

APPENDIX B

Code snippets and configuration files

```
# 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.1: Raspbian configuration for static ip address

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

Listing B.2: Raspbian configuration for dynamic ip address

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

all: clean
	gcc $(CFLAGS) master.c sec.c clr.c knx.c cryptoHelper.c -o
	    master -pthread $(LIBS)

debug: clean
	gcc $(CFLAGS) master.c clr.c sec.c knx.c cryptoHelper.c -o
	    master -pthread -DDEBUG $(LIBS)

clean:
```

```
12          clear
13          rm -rf *.o
14          rm -f master
15
16 drun: debug
17          @echo "starts master daemon in debug mode, with KNX device
                addr 1"
18          ../start.sh 1
19
20 run: all
21          ../start.sh 1
22
23 update:
24          git commit -a --allow-empty
25          git pull
26          git push
```

Listing B.3: Makefile for the master daemon

Glossary

3DES Triple Data Encryption Standard. 20–22

ACU Advanced Coupler Unit. 63

AE Authenticated Encryption. 2, 28

AES Advanced Encryption Standard. 22, 31, 63, 64, 90, 91, 93, 94

AIL Application Interface Layer. 62, 63

APDU Application Layer Protocol Data Unit. 60, 63

API Application Programming Interface. 82, 88, 90, 98

BbC Backbone Coupler. 55

bps bits per second. 55, 79

CA Certification Authority. 64

CA Collision Avoidance. 56

CBC Cipher Block Chaining. 23–25, 29–32, 63

CCM Counter with CBC MAC. 31, 63

CFB Cipher Feedback Mode. 25, 26

CPA Chosen Plaintext Attack. 24

CRC Cyclic Redundancy Check. 28, 64

CSMA Courier Sense Multiple Access. 56

CTR Counter Mode. 24, 26, 31, 32, 63, 64, 90

CTRLE Extended Control Field. 57

D-H Diffie-Hellman. 12, 14, 34–37, 71, 72, 92

DANP Doubly Attached Node with PRP. 51

DES Data Encryption Standard. 20–22

DLP Discrete Logarithm Problem. 12, 14, 35

DOS Denial of Service. 1, 2, 45, 91

DPT Data Point. 53

DSA Digital Signature Algorithm. 39

DT Data Types. 53, 60

EC Elliptic Curve. 12, 14–16, 35

ECB Electronic Code Book. 23, 24

ECC Elliptic Curve Cryptography. 34, 35, 39

ECDH Elliptic Curve Diffie-Hellman. 71, 77, 84, 85, 88, 89, 92

ECDLP Elliptic Curve Discrete Logarithm Problem. 16, 36, 39, 92

ECDSA Elliptic Curve Digital Signature Algorithm. 39

EHS European Home Systems Protocol. 53

EIB European Installation Bus. 53–55

EIBD European Installation Bus Daemon. 82–84, 98

EIS EIB interworking standard. 53

ETS Engineering Tool Software. 53, 63, 64

FDSK Factory Default Setup Key. 63

GA Group Address. 58, 59, 67–69, 72, 73, 77, 80

HA High Availability. 43, 45

HBA Home and Building Automation System. 1, 53, 61, 62

HMAC Keyed-Hash Message Authentication Code. 29, 90

HSR High Availability Seamless Redundancy. 51, 52

HVAC Heating, Ventilation and Air Conditioning. 1, 61

IA Individual Address. 56, 58, 59, 67–69, 72, 73, 77, 79, 80

IEEE Institute of Electrical and Electronics Engineers. 48, 50

IP Internet Protocol. 1, 51, 55, 62, 64, 65

IPSec Internet Protocol Security. 65

IPv4 Internet Protocol version 4. 59, 65

ISO International Organization for Standardization. 53

IV Initialization Vector. 11, 23–26, 32, 63

KNX Konnex. 1–3, 53–56, 59, 60, 62–70, 72–74, 77–80, 82, 83, 85, 91, 94

LAN Local Area Network. 51

LC Line Coupler. 55

LFSR Linear Feedback Shift Register. 18, 19

LLC logical link control. 56

LRE Link Redundancy Entity. 51

LSDU Link Service Data Unit. 51, 57

MAC Message Authentication Code. 8, 23, 27–29, 31, 32, 38, 63, 70, 71, 76–78, 80, 90, 92–94

MAC Medium Access. 31, 49, 51, 56

MAU Medium Access Unit. 56

MRP Media Redundancy Protocol. 50

MTTF Mean Time To Failure. 41

MTTR Mean Time To Repair. 41, 42

NIST National Institute of Standards and Technology. 20, 22, 29, 39, 88

NSA National Security Agency. 21

OFB Output Feedback Mode. 26

OSI Open System Intercommunication Model. 48, 51, 53, 54, 59, 65

OTP One Time Pad. 17

PDU Protocol Data Unit. 63

POSIX Portable Operating System Interface. 85, 87

PRNG Pseudo Random Number Generator. 11, 12, 18

PRP Parallel Redundancy Protocol. 51, 52

PSK Pre Shared Key. 64, 77

RAID Redundant Arry of Independent Disks. 45

RF Radio Frequency. 55

RSTP Rapid Spanning Tree Protocol. 50

S-AL Secure-Application Layer. 62, 63

SAN Single Attached Node. 51

SHA Standard Hashing Algorithm. 29, 90

SLA Service Level Agreement. 41

SP Substitution-Permutation. 20

SPOF Single Point of Failure. 65

SSH Secure Shell. 28, 31, 96

SSL Secure Sockets Layer. 31

STP Spanning Tree Protocol. 48–50

TCP Transmission Control Protocol. 55

TCP Transport Control Protocol. 1, 59, 64

TK Tool Key. 63

TLS Transport Layer Security. 64, 65

TMR Tripple Modular Redundancy. 46

TP Twisted Pair. 3, 54–56, 91

TPCI Transport Layer Protocol Control Information. 59, 63

TPDU Transport Layer Protocol Data Unit. 59

TRNG True Random Number Generator. 12

TTL Time To Live. 59

URL Uniform Resource Locator. 98

USB Universal Serial Bus. 2, 98, 99

VLAN Virtual Local Area Network. 49

VPN Virtual Private Network. 62

XOR Exclusive-Or. 17, 18, 20–22, 24, 25, 32, 33

Bibliography

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

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

[3] W. Stallings. "Computer Security". In: New Jersey: Pearson Education International, 2008. Chap. Overview, p. 15.

[4] D. Kahn. "The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet". In: New York: Scribner, 1996.

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

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

[7] S. Goldwasser and S. Micali. "Probabilistic Encryption". In: *Journal of Computer and System Science* (1984), pp. 270–299.

[8] C.E. Shannon. "A mathematical theory of communication". In: *Bell System Technical Journal, The* 27.3 (July 1948), pp. 379–423. ISSN: 0005-8580.

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

[10] 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.

[11] 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*. Apr. 2010. URL: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf.

[12] S.P. Dwivedi. “Computing multiplicative order and primitive root in finite cyclic group”. In: *Contemporary Computing (IC3), 2014 Seventh International Conference on*. Aug. 2014, pp. 130–134. DOI: 10.1109/IC3.2014.6897161.

[13] Jim Zhang and Li-Qun Chen. “An Improved Algorithm for Discrete Logarithm Problem”. In: *Environmental Science and Information Application Technology, 2009. ESIAT 2009. International Conference on*. Vol. 2. July 2009, pp. 658–661. DOI: 10.1109/ESIAT.2009.457.

[14] D Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptopgraphy*. New York: Springer, 2004, p. 153.

[15] 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.

[16] J. Menezes, P. van Oorschot, and Vanstone S. “Handbook of Applied Cryptography”. In: New York: CRC Press, 1997. Chap. Overview of Cryptography, p. 196.

[17] H. Feistel. “Cryptography and Computer Privacy”. In: *Scientific American* 228.5 (May 1973), pp. 15–23.

[18] H. Feistel. *Block cipher cryptographic system*. US Patent 3,798,359. Mar. 1974. URL: http://www.google.com/patents/US3798359.

[19] National Institute of Standards and Technology. *Data Encryption Standard @ONLINE*. 1979. URL: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[20] National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation @ONLINE*. 2001. URL: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[21] K. Burda. “Error Propagation in Various Cipher Block Modes”. In: *International Journal of Computer Science and Network Securit* 6.11 (Nov. 2006), pp. 235–239.

[22] N. Ferguson, B. Schneier, and T. Kohno. “Cryptography Engineering”. In: Indianapolis: Wiley, 2010. Chap. Block Cipher Modes.

[23] M Zalewski. *SSH CRC-32 compensation attack detector vulnerability (2001)*.

[24] Danilo Gligoroski et al. “Internationally standardized efficient cryptographic hash function”. In: *Security and Cryptography (SECRYPT), 2011 Proceedings of the International Conference on*. July 2011, pp. 426–433.

[25] J. Menezes, P. van Oorschot, and Vanstone S. “Handbook of Applied Cryptography”. In: New York: CRC Press, 1997. Chap. Overview of Cryptography, p. 329.

[26] National Institute of Standards and Technology. *Secure Hash Standard @ONLINE*. Mar. 2012. URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[27] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding Collisions in the Full SHA-1". In: *In Proceedings of Crypto*. Springer, 2005, pp. 17–36.

[28] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. RFC Editor, Feb. 1997. URL: http://tools.ietf.org/html/rfc2104.

[29] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. "Breaking and Provably Repairing the SSH Authenticated Encryption Scheme: A Case Study of the Encode-then-Encrypt-and-MAC Paradigm". In: *ACM Trans. Inf. Syst. Secur.* 7.2 (May 2004), pp. 206–241. ISSN: 1094-9224.

[30] Serge Vaudenay. "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ..." In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. 2002, pp. 534–546.

[31] W. Stanley Jevons. *The Principles of Science"*. London, 1913, p. 144.

[32] Ralph C. Merkle. "Secure Communications over Insecure Channels". In: *Commun. ACM* 21.4 (Apr. 1978), pp. 294–299. ISSN: 0001-0782.

[33] Boaz Barak and Mohammad Mahmoody-ghidary. "Merkle puzzles are optimal: An O(n 2)- query attack on key exchange from a random oracle". In: *In Advances in Cryptology — Crypto 2009, volume 5677 of LNCS*. Springer, 2009, pp. 374–390.

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

[35] N. Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of Computation* 48.177 (1987), pp. 203–209.

[36] Victor S Miller. "Use of Elliptic Curves in Cryptography". In: *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*. New York, NY, USA: Springer-Verlag New York, Inc., 1986, pp. 417–426.

[37] D Hankerson, A. Menezes, and S. Vanstone. *Guid to Elliptic Curve Cryptopgraphy*. New York: Springer, 2004, p. 172.

[38] 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.

[39] J. Jonsson and B. Kaliski. *Public-Key Cryptography Standards (PKCS)1: RSA Cryptography*. RFC 3447. RFC Editor, Feb. 2003. URL: http://tools.ietf.org/html/rfc3447.

[40] "IEEE Standard Specifications for Public-Key Cryptography". In: *IEEE Std 1363-2000* (Aug. 2000), pp. 1–228. DOI: 10.1109/IEEESTD.2000.92292.

[41] National Institute of Standards and Technology. *DIGITAL SIGNATURE STANDARD @ONLINE*. July 2013. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[42] Arjen K. Lenstra. *Key Length*. 2004.

[43] J.C. Knight. "Safety critical systems: challenges and directions". In: *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. May 2002, pp. 547–550.

[44] C.W. Axelrod. "Applying lessons from safety-critical systems to security-critical software". In: *Systems, Applications and Technology Conference (LISAT), 2011 IEEE Long Island*. May 2011, pp. 1–6.

[45] F.V. Brasileiro et al. "Implementing fail-silent nodes for distributed systems". In: *Computers, IEEE Transactions on* 45.11 (Nov. 1996), pp. 1226–1238.

[46] J. von Neumann. "Probabilistic logics and synthesis of reliable organisms from unreliable components". In: *Automata Studies*. Ed. by C. Shannon and J. McCarthy. Princeton University Press, 1956, pp. 43–98.

[47] R. E. Lyons and W. Vanderkulk. "The Use of Triple-modular Redundancy to Improve Computer Reliability". In: *IBM J. Res. Dev.* 6.2 (Apr. 1962), pp. 200–209.

[48] R. Pallos et al. "Performance of rapid spanning tree protocol in access and metro networks". In: *Access Networks Workshops, 2007. AccessNets '07. Second International Conference on*. Aug. 2007, pp. 1–8.

[49] G. Prytz. "Redundancy in Industrial Ethernet Networks". In: *Factory Communication Systems, 2006 IEEE International Workshop on*. 2006, pp. 380–385.

[50] A. Giorgetti et al. "Performance Analysis of Media Redundancy Protocol (MRP)". In: *Industrial Informatics, IEEE Transactions on* 9.1 (Feb. 2013), pp. 218–227.

[51] H. Kirrmann, M. Hansson, and P. Muri. "IEC 62439 PRP: Bumpless recovery for highly available, hard real-time industrial networks". In: *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. Sept. 2007, pp. 1396–1399.

[52] J.A. Araujo et al. "Duplicate and circulating frames discard methods for PRP and HSR (IEC62439-3)". In: *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*. Nov. 2013, pp. 4451–4456.

[53] S.A. Nsaif and Jong Myung Rhee. “Improvement of high-availability Seamless Redundancy (HSR) traffic performance”. In: *Advanced Communication Technology (ICACT), 2012 14th International Conference on*. Feb. 2012, pp. 814–819.

[54] 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. Sept. 2013, pp. 31–39.

[55] KNX Association. “Systems Specifications, Communication Medium TP1”.

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

[57] W. Kastner et al. “Communication Systems for Building Automation and Control”. In: *Proceedings of the IEEE* 93.6 (June 2005), pp. 1178–1203.

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

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

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

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

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

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

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

[65] 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*. Sept. 2010, pp. 1–4.

[66] 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*. Nov. 2009, pp. 2459–2464.

[67] 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.

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

[69] 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.

[70] Hugo Krawczyk. *The order of encryption and authentication for protecting communications (Or: how secure is SSL?)* Cryptology ePrint Archive, Report 2001/045. http://eprint.iacr.org/. 2001.