TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

Security in Networked Building Automation Systems

ausgeführt am

Institut für Rechnergestützte Automation
Arbeitsgruppe Automatisierungssysteme
der Technischen Universität Wien

unter der Anleitung von

ao. Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Kastner
und
Univ.-Ass. Dipl.-Ing. Georg Neugschwandtner

durch

Wolfgang Granzer
Waidhofnerstraße 15
3331 Hilm

Wien, 14.10.2005

Kurzfassung

Systeme der Gebäudeautomation (GA) beschäftigen sich mit der automatischen Steuerung von gebäudetechnischen Subsystemen. Darunter fallen beispielsweise Anlagen zur Steuerung von Heizung, Lüftung und Klima (HLK). Zugriffsschutz und -sicherheit waren in der GA bislang unterbewertete Themen, da man davon ausging, dass der Angriff auf ein bestehendes Subsystem ohne physikalischen Zugang nicht möglich ist. Durch die Anbindung von GA an das Internet ist dieser physikalische Zugang nunmehr jedoch nicht mehr notwendig.

Aus diesem Grund ist die vorliegende Diplomarbeit dem Zugriffsschutz und der Zugriffssicherheit in der Gebäudeautomation gewidmet. Unterschiedliche Konzepte und Mechanismen, die zum Schutz und Sicherheit von Systemen dieses Bereiches herangezogen werden können, werden einleitend vorgestellt. Potentielle Szenarien für Bedrohungen und Angriffspunkte werden identifiziert sowie entsprechende Gegenmaßnahmen diskutiert.

In weiterer Folge werden bestehende Netzwerke der Gebäudeautomation und deren integrierte Sicherheitskonzepte analysiert. Besonderes Augenmerk wird auf ein weit verbreitetes System namens EIB/KNX gelegt. Da die Sicherheitskonzepte in EIB/KNX nur rudimentär vorhanden sind, wird eine Erweiterung namens EIBsec vorgestellt. EIBsec unterstützt Datenintegrität, Vertraulichkeit, Aktualität und Authentifizierung. Weitere Kernpunkte, wie Schlüsselverwaltung und automatische Verteilung von Software Patches, sind ebenso Bestandteil von EIBsec.

Abstract

Building Automation Systems (BAS) are concerned with automated control of building subsystems such as Heating Ventilation and Air Conditioning (HVAC). In this domain security issues have been underrated over the past years, since physical access to the subsystem of interest was typically mandatory for an attack. However, especially since the integration of BAS with the Internet, things have changed.

This thesis gives a survey on security in building automation systems. After a brief introduction, different security concepts and mechanisms are presented. Next, possible threats, attacks and available countermeasures are discussed.

Furthermore, the security mechanisms of currently available networks for BAS are analysed. Finally, an extension to a popular building automation network is presented. EIBsec extends EIB/KNX to support several security mechanisms that guarantee data integrity, confidentiality and freshness, as well authentication. Issues such as key management and distribution of software updates are also addressed.

Danksagung

An dieser Stelle möchte ich mich bei allen Menschen bedanken, die mich während meines gesamten Studiums und besonders während der Verfassung dieser Arbeit unterstützt haben.

Besonderen Dank gilt dabei meinen Eltern und Großeltern, die mich während des gesamten Studiums immer tatkräftig zur Seite standen.

Weiters möchte ich meinem Betreuer Wolfgang Kastner sowie Georg Neugschwandtner danken, ohne die diese Arbeit nicht denkbar gewesen wäre. Zusätzlich möchte ich noch Christopher Kruegel dafür danken, dass er sich die Zeit nahm, den Security Teil meiner Arbeit auf seine Richtigkeit zu prüfen.

Außerdem möchte ich noch bei meinen Freunden Jagal und Mats dafür bedanken, dass sie sich trotz der knappen Zeit bereit erklärt haben, diese Diplomarbeit korrektur zulesen.

Bei meinen Studienkollegen, besonders bei Fritz, Beno und Gerd, bedanke ich mich für die Freundschaft, den Spaß und die gegenseitige Unterstützung.

Außerdem möchte ich mich noch bei Michaela bedanken, die mich während der gesamten Arbeit unterstützte.

Contents

Chapter 1

Introduction

1.1 Motivation

Building automation systems have their historical root in a time where security was a neglected topic. Due to the fact that security threats were rare, no special security mechanisms were implemented. Therefore, available solutions provide no or only rudimentary protection against malicious attacks.

In other IT domains, the situation is very similar. Consider, for example, Internet-related protocols like TCP/IP, POP and FTP. These protocols were developed in a time where nobody thought about security. The developers did not consider security as an important issue and therefore security countermeasures were not implemented. Nowadays, the Internet is flooded with viruses, trojan horses and worms. Since the used protocols cannot avoid these security threats, security extensions (like SSL) and special software programs (like anti-virus or firewalls) became necessary.

In the past, building automation systems were isolated. An interconnection to other foreign networks was typically uncommon and therefore ”security by obscurity” was mostly sufficient. With the spreading of the Internet, many building automation systems began to support an interconnection to other public WANs. This interconnection provides new opportunities for potential attackers. Since a physical access to the internal network is no longer needed, it is easier to attack the building automation system from the outside. Therefore, it is necessary to protect these interconnection points against malicious attacks.

Even if the building automation system is isolated from other systems, there are still possibilities to gain unauthorised access from the inside. It is also im-

portant to note that a protection against security attacks from foreign networks is typically not enough. If an attacker bypasses this ”single wall of protection”, the attacker has full access to the internal network. Therefore, it is essential to protect the building automation system against security threats from the inside too.

Hence, security is getting more and more important in the building automation domain. Therefore, this thesis discusses security in building automation systems. It also presents security extensions for a popular building automation network protocol.

1.2 Guide through the Thesis

The main aim of this thesis is to give an overview about security in building automation systems. First, Chapter 2 defines what a building automation system is. Additionally, basic terms and definitions are explained which are necessary for the rest of this thesis. In Chapter 3, basic security concepts and mechanisms are explained. In addition to a detailed description, the importance of these mechanisms for the building automation domain is discussed. The next chapter (Chapter 4) shows how a malicious user is able to attack a building automation system. While Chapter 4 gives an overview about security threats and countermeasures, the next Chapter 5 analyses the security architecture of available building automation solutions. Finally, the last chapter (Chapter 6) takes a closer look at EIB/KNX. The security mechanisms in EIB/KNX are analysed and at the end of Chapter 6, a possible secure EIB/KNX architecture called EIBsec is presented.

Chapter 2

Building Automation Systems

This chapter contains important definitions and explanation of terms which are necessary for the study of security aspects of building automation systems.

2.1 What is a Building Automation System?

It is important to explain what automation of a building means. Depending on the purpose of the building, the people who work in the building or the machines, processes and produced goods have different demands. To fulfil these demands, a technical infrastructure is necessary. This technical infrastructure is known as **building services** ([1]). To control these building services, different control systems are needed.

As mentioned in [1], building automation is concerning the control of building services. Therefore, a control system which provides automatic control of building services is called a **building automation system**. The goal of such a building automation system is to make a building as ”intelligent” as possible. Due to this automatic control, such buildings are often referred to as **intelligent buildings**. In earlier days, this automation was often achieved in a centralised way. Centralised means that automatic control is done by a single controller or control station.

Nowadays, computer networks are getting more and more important. A lot of research is done into network technology. In the field of building automation, a trend towards the use of network technologies can be observed. Due to the disadvantages of centralised solutions (performance bottle necks, single point of failure, ...), a distributed model is often more practical. A distributed approach needs a network which can be used by the different devices to communicate with

each other. Such a network is applied to a building automation design it is also referred to as a **building automation network**.

A building automation system is a special type of an industrial automation system (for further details see [1]). Therefore, a lot of properties are very similar. It is important to note that there are still differences between a building automation system and an industrial automation system. As these differences can affect the system behaviour significantly, they must be carefully taken into account.

Automatic control has a lot of advantages. First, the use of a building automation system makes it easier to maintain the systems in the building. Additionally, it is possible to modify the system behaviour even if the installation of the system has been finished. For example, in an intelligent building it is possible to change the configuration of the lighting system without changing the physical wiring. The system operator can simply reassign the binding between the light switch and the ceiling light without changing the electrical wiring.

Another advantage of the use of a building automation system can be the improvement of the energy efficiency. With such systems, it is possible to take the relationship of different systems (e.g. heating, ventilation and air conditioning) into account. Therefore, optimisations are possible. For example, if somebody opens the window the control system can turn off the heating for this room.

Additionally, with such systems it is possible to control and monitor the systems in a building from a centralised control center. Such a control center provides a global view of the whole system. So, for the system operator it is easier to monitor and maintain the building automation system. Especially, in a large building such a centralised approach can reduce the costs of maintenance (for example because of the global view, a system error can be detected and corrected easier). Furthermore, the building automation system can be maintained remotely (for example through a HTML interface from the internet).

Disadvantages on the other hand are that the construction costs of a building automation system are higher. Additionally, the employees who maintain the building automation system must be familiar with the automation system. Therefore, the system operators must be well-trained. Sometimes further training is necessary. Due to the high lifetime of buildings, the operational costs are much higher than the installation costs. Therefore, it is more practicable to use building automation systems.

It is important to distinguish between the automation of large buildings and smaller ones (for example homes). Large buildings (for example large office buildings, warehouses, schools, churches, ...) are more complex and have hundreds or even thousands of devices. Due to these large number of devices, building

systems are harder to administrate and harder to maintain. Therefore, the use of a building automation system is more economic. Due to the decreasing installation costs, the automation of smaller buildings like homes becomes more important recently. Nowadays, it is not uncommon to integrate a building automation system in single-family houses.

2.2 Systems in a Building

Depending on the demands of the building, a building automation system must control different types of building services. Based on [1], this thesis distinguishes between four different types of systems in a building. These four different types are shown in Figure 2.1.

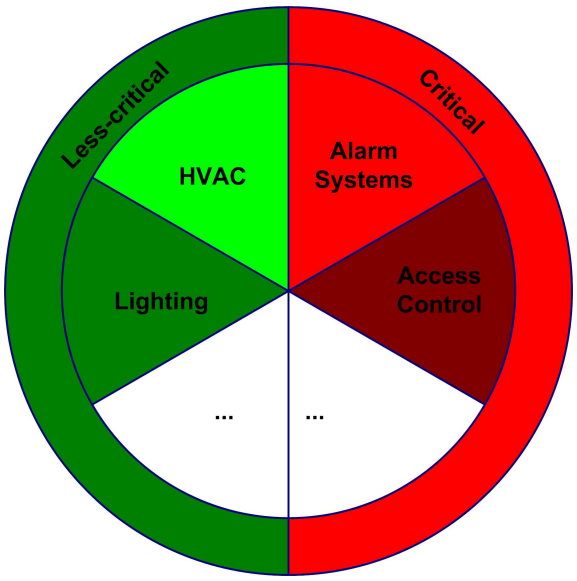


Figure 2.1: Different types of systems

As mentioned in Figure 2.1, a system can be classified as less critical or as critical. To explain the difference between critical and less critical systems, it is important to define the terms failure, fault and error. A **failure** is an event that denotes a deviation between the actual service and the specified one ([22]). For example, an incorrect sensor value is a failure. An **error** denotes an incorrect

internal state ([22]). For example, a wrong data element is an error. The cause of an error, and therefore the indirect cause of a failure is called a **fault** ([22]). For example, such a fault can be a broken wire (physical fault) or an implementation flaw (design fault). To avoid such faults, the system must be safe and secure (for further details about failures, faults and errors and their classification see[22]).

The classification into critical and less critical systems depends on the damage, which could be caused by a system failure. There are systems where a failure and its failure costs are not that critical. For example, if a light is broken, it will not result in a catastrophe. Such a system is classified as a **non critical system**. It is important to note that this classification depends on the environment too. Normally, a broken light will not end in a catastrophe. For example, in an operating room the situation is different. If the light fails, this could be a disaster. It is obvious that such a lighting system should be classified as a critical system. Or if the whole lighting system of a cinema fails, the visitors could panic. Additionally, the cinema must be closed until the lighting is repaired. If the cinema must be closed, this will cause a lot of economical costs for the cinema. Again, the term non critical would be unsuitable for such a system. Therefore, a building automation system cannot strictly be classified as critical or non critical. It depends on the building type and the particular building service which should be served. Generally, the term non-critical is impractical. Since every system has its critical part, it is better to use the term **less critical** instead.

As shown in Figure 2.1, an **HVAC** (heating, ventilation and air conditioning) system is classified as a less critical system. HVAC systems are the most common type of building systems. An HVAC control system regulates the room temperature and controls the air flow. The main aim of such a control system is to provide a comfortable environment. The human needs of the people in the building should be satisfied as good as possible. To get the relevant information, sensors (both indoor and outdoor) are needed. Additionally, the dependencies between these values must be considered too. For example, the air flow and the humidity influence the room temperature too. It is important to note that there are a lot of variables which cannot be measured. For example, every person has a different sensation about temperature which cannot be directly measured with sensors. Even the clothing of the people influences this sensation and therefore the control of the HVAC system too. So, the controlling of an HVAC system is not a trivial task. Automatic control of an HVAC system also helps to save energy. To achieve this, it is important to exchange information between different control systems. For example, the solar radiation and the lighting system influence the temperature too. It is important that these energy saving mechanisms do not decrease the user

comfort. The goal is to maximise the user comfort and to decrease energy costs at the same time.

The **lighting system** is another typical less critical system. Two different types of lighting systems exist. The **artificial lighting** is responsible for switching and dimming luminaires. Additionally, the lighting can be controlled automatically without manual operations. Sensors can be used to detect the presence of people. If somebody enters the room, the light is turned on automatically. This scheme does not only increase the user comfort. It can also save energy. For example, the lighting system can be configured to turn off the light automatically, if nobody is in the room.

The lighting system may not only consist of switches and lights. A **daylighting** system regulates the amount of daylight which enters the building. To achieve this, blinds are used to control the shading of the building.

If a system failure can cause damage, the system is classified as a **critical system**. A critical system should be safe and secure. To achieve these requirements, critical systems should be fault tolerant and fail safe. This means that if an unavoidable fault occurs, only a part of the system should be affected. Additionally, the consequences and the damage should be as little as possible. Furthermore, it should be guaranteed that the system enters a fail safe state where no catastrophe will result. Due to the relationship between security and safety, a critical system should be protected against security attacks too.

It is important to distinguish between safety and security critical systems. The main difference between safety and security is the type of protection. **Safety** measures reduce the risk against unintended system states which can cause damage. For example, a fire alarms system is a typical safety system. **Security** measures protect the system (the building) against malicious attacks caused by humans. For example, an intrusion detection system is a typical security system. Therefore, the fault nature is the main difference between safety and security. For the rest of this thesis, a maliciously introduced fault is called a **security attack**. The term **fault** is used to denote an unintended fault which is not a malicious attack.

Security and safety issues are closely related. The security of a building should be guaranteed, even if the building itself is not safe anymore. Consider, for example, a large building that has a vault. Normally, a fire alarm will open all emergency doors of a building. If a part of the building catches fire (even if the cause is not a malicious attack), the security alarm system of the building must not fail. It is important that only the doors of the affected part of the building will be opened. Otherwise, an intruder could raise an alarm in an unprotected part of the building (for example, a room which is easily accessible) and prompt the alarm

system to open the emergency doors to the vault. It is also important that a safety system must not shutdown, if someone attacks the building automation system. If someone tries to gain unauthorised access to the building, this security attack must not shutdown safety systems. Consider, for example, an access control system of a laboratory which can only be entered with protective clothing. It is essential that the access control system is still working even if a security attack happens. Otherwise, the safety of the system cannot be guaranteed anymore. So, a safety system like the access control system mentioned above can only guarantee safety in all unpredictable situations, if the safety system itself is secure enough.

The alarm system is a typical critical system. An **alarm system** monitors the state of the building and compares the current state with a reference one. If there is an unexpected difference between these states, the system raises an alarm or initiates other alarm actions. The most important goal of these systems is to distinguish clearly between alarm and non-alarm situations. As mentioned before, fault tolerance and security are the most important requirements.

Depending on the state which should be monitored, different types of alarm systems exist. On the one hand, there are so called **safety alarm systems**. Fire alarm, water leakage and smoke detection systems are typically safety alarm systems.

On the other hand, **security alarm systems** protect and monitor the building against unwanted intrusion and damage caused by humans. Typical security alarm systems are intrusions detection (e.g. glass break sensors, audio surveillance,...) and motion tracking systems. It is obvious that the control system itself must be secure. The security of a building can only be guaranteed if the control system itself is secure enough. For example, if a burglar is able to disable the sensors and cameras of the intrusions detection system, the security of the building cannot be guaranteed. Even, if there are thousands of sensors and cameras, the system itself must be protected against security attacks. Otherwise, the intruder could bypass the security system.

An **access control** system is a special type of a critical system. The main aim of these systems is to gain or deny the access to restricted areas. Again, two different types of access control systems exist.

On the one hand, a **security access control system** has to verify the identity of the persons who want to gain access. If the identity has been successfully proved, the privileges of the person for the requested area are checked. If the person has the required access rights, the access will be granted (otherwise it will be denied). With such systems it is not only possible to make simple ”access granted/access denied” decisions. Depending on the privileges of the person, it is possible to

restrict the access to particular areas. To achieve this, the elevator of a building could be controlled by the access control system. For example, office employees are only allowed to gain access to the office floors. Other floors like the research center are restricted to employees who have the necessary rights. Like the security alarm system, the access system itself must be secure. To avoid unauthorised access, it must be prohibited that someone bypasses the access control mechanism or that someone fakes the identity of an authorised person.

On the other hand, a **safety access control system** verifies whether it is safe enough to grant access to a particular part of the building. A typical example of a safety access control system is a chemical laboratory. The access control system of such a laboratory has to verify whether the user satisfies the required safety requirements. For example, it could be necessary to wear a protecting suit to enter the laboratory safely. The access control system must guarantee that only employees that wear such a protecting suit are allowed to gain access to the laboratory.

In the past, the control mechanisms of these building services were implemented as separate systems. These different systems were isolated and therefore no interaction was possible between these systems. Nowadays, interaction between different systems is getting more popular. With the combination of different systems, the whole building automation system becomes more flexible and the possibilities of such systems have improved. For example, if the access control system is combined with the lighting system, it is possible to highlight the way through the building which the person has to go.

This interaction increases the complexity of the automation system. This means that interaction points must be carefully designed. Therefore, these points should be limited to a necessary amount.

It is often not allowed to combine life safety systems like the fire alarm system with other building automation system. To reduce the error probability, such critical systems must be implemented as an isolated, independent system. If in future a separation is not needed anymore, it could be possible that all different building services are controlled with one ”All-in-One building automation system”. In order to achieve this, these systems must be fault tolerant and secure enough.

2.3 Network Topology

The standard system model for building automation systems is shown in Figure 2.2. For our purposes, this model has been simplified (for further details see [13]).

In this model, the system functionality is divided into three levels. These three different functionality levels are ordered hierarchically.

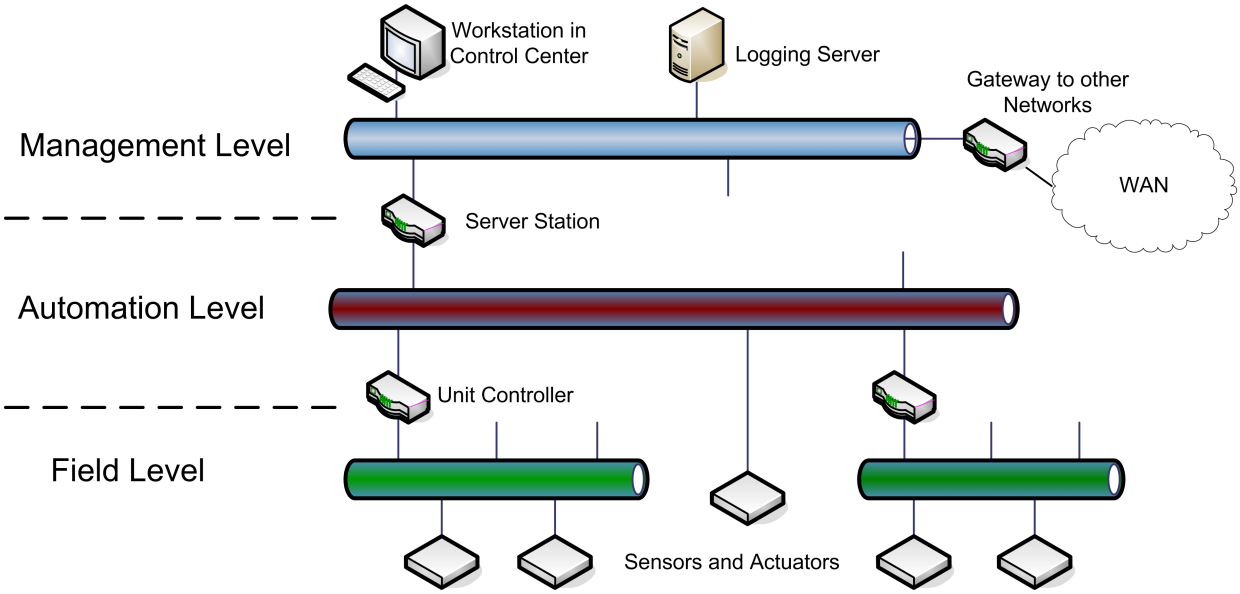


Figure 2.2: Three-level functional hierarchy

The **field level** is the lowest one. This level directly interacts with the physical environment. The main aim of this level is to collect the necessary data, transmit data to the automation level and act upon the environment.

The collected data is transmitted to the **automation level**. Automatic control is done in this level. This means that the data which is collected at the field level is processed. Depending on the entities which are involved, two different types of communication are possible. **Horizontal communication** means the process data is exchanged between different processing entities. Additionally, the automation level also prepares the data values for the management level. If data is transmitted to the management level, this type of communication is called **vertical communication**.

The **management level** provides a global view of the whole system. This level is responsible for manual controlling and logging. The operator of the system has the possibility to configure and to monitor the building automation system (for example from a workstation in the control center). Additionally, the management level is responsible for collecting statistical data (logging) and archiving it. Inter-

connections to other networks are also possible at this level (for example to other WANs or to an office LAN).

Normally, automation systems are designed distributed. A distributed scheme has a lot of advantages like better fault tolerance (no single point of failure), less bottlenecks, less latencies in control loops and loose coupling (if one node fails only a part of the system is affected). It is important to note that distributed systems are more complex and therefore harder to design. It can also be observed that at higher levels the amount of data which must be transmitted increases. Compared to the packets at the management level, the data packets at the field level are very small (often only a few bytes). Therefore, the required bandwidth is significantly smaller than the bandwidth at higher levels. For example, at the management level LANs and WANs with several MBits/sec or even GBits/sec are used whereas field bus networks have a few KBits/sec. It is important to note that time constraints become more important at the field level (for further details see Section 2.4).

As mentioned before, each level of this functional hierarchy represents a part of the system functionality. Each device of the building automation system implements a particular function. Today, the trend goes towards so called **intelligent devices**. In addition to the standard functionality, an intelligent device performs additional tasks too. In terms of building automation, such an intelligent device implements functionality from more than one level. For example, a field device like a sensor often performs automation tasks too (intelligent sensor). Therefore, this three level functional hierarchy is implemented as a flatter, two-level architecture (see Figure 2.3). This two-level model consists of a control network level and a common backbone. This thesis will take this two-level model as basis for further considerations.

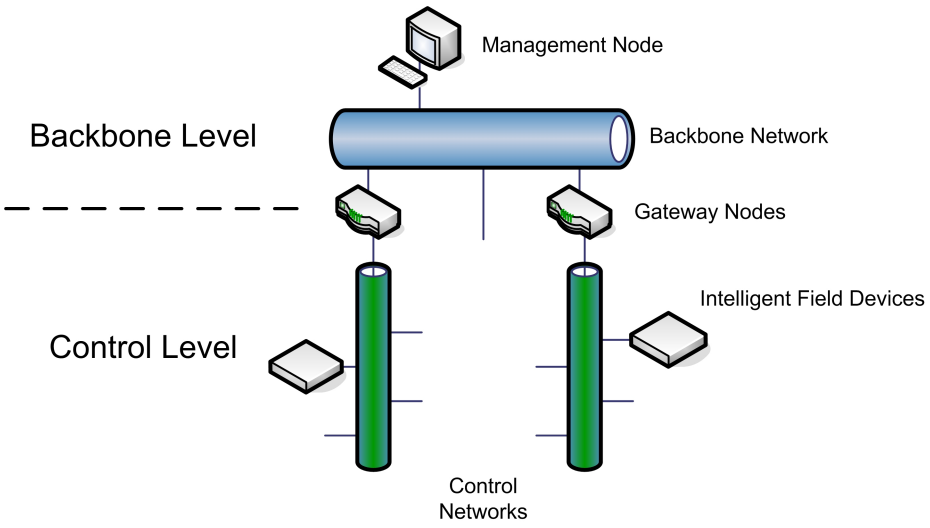


Figure 2.3: Two-level Model

The **control network** consists of intelligent sensors and actuators. Compared to an ordinary sensor or actuator, such an intelligent device implements more functionality. Therefore, a separate automation network is often not necessary. It is important to note management devices like logging servers or maintenance workstations still have requirements which cannot be fulfilled by this control network. For example, typical control networks do not have the necessary network bandwidth to meet the throughput requirements of the management devices which are needed.

Therefore, the different control networks are interconnected by a **backbone network**. This backbone has the necessary bandwidth and performance to satisfy these needs. With this approach, it is not only possible to integrate the necessary management nodes. The backbone network can also link different control networks together. It is even possible to interconnect control networks that use different network protocols.

At the intersection point between the control network and the backbone, **interconnection devices** (for further details see Section 2.5) are used. These devices are responsible for routing (router) the traffic between the different networks as well as translating (gateway) the network messages (for further details see Section 2.5). Therefore, each interconnection device contains a database which contains the mapping table. Additionally, a gateway can perform other tasks too. For exam-

ple, a gateway could provide an HTML interface which can be used to configure the gateway or to monitor the network traffic. A gateway could act as a logging device, too. Since these interconnection devices are installed at critical points of the network, they must be fault tolerant and carefully protect against security attacks.

Depending on the amount of network segments that the interconnection device connects, it has two or more network interfaces. If a device transmits a data packet to a device that is located in a foreign control network, the interconnection device is responsible for routing this packet to the interconnection device of the destination control network. In this thesis, this type of communication is referred to as **bottom-up communication**.

At the backbone side, different management devices establish connections to the interconnection device to perform management tasks. For example, a logging server connects to the different gateways to retrieve statistical data. This type of communication is called **top-down communication**.

If an interconnection device interconnects two different networks (with different network protocols), the required network services must be mapped on services of the foreign network. As these services are very protocol-specific, a mapping of the services is hard or even impossible. Since the effort to do such a mapping is very high, a scheme called **tunneling** is often used (see Figure 2.4).

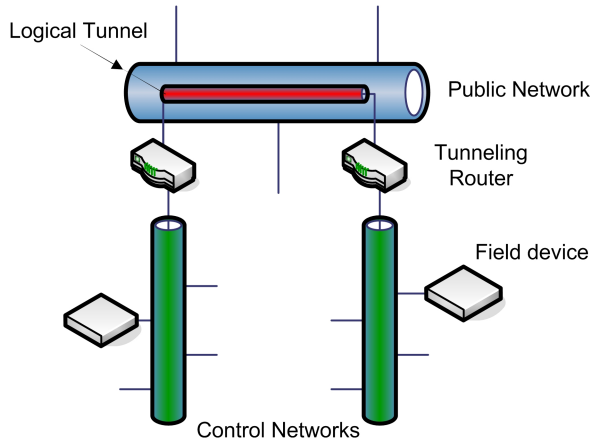


Figure 2.4: Tunneling

Tunneling means the whole data packet is transmitted through a logical tunnel. With this approach, the packet from the control network remains untouched.

Untouched means the protocol specific data (header, trailer) of the packet and the data itself is not modified. The data packet is put into a ”backbone packet” and then it is transmitted through the tunnel to the destination tunneling router. It is like putting a letter into an envelope. The destination router unpacks the packet and the untouched packet is sent to the destination device. With this scheme, it is possible to use networks as backbone which cannot provide all required network services. Today, nearly every bigger building (especially office buildings) has an IP network installed. Therefore, it is very popular to use the already existing IP network as backbone network. So, no additional network has to be installed.

A disadvantage with the tunneling approach is the control networks are not decoupled. Due to the transparent transmission, the different networks stay dependent. Using gateways, the systems would be decoupled. Therefore, real time constraints between networks which are connected through a logical tunnel are often harder to guarantee.

Normally, control networks and backbone networks are designed for different needs. Since maintenance and administration are performed at single points in the network (for example a central logging server), the backbone network needs more bandwidth than the control network for example. So, these differences between the control network and the backbone must be taken into account (for further details see Section 2.4).

2.4 Control Network

2.4.1 Demands on the Protocol

As mentioned in Section 2.3, compared to the backbone network the network protocol of the control network has to meet different demands.

First, the **bandwidth** requirements are different. At the control level the transmitted packets are smaller (often only a few bytes). Therefore, less bandwidth is needed (even a few KBits/sec are often enough).

At the control level, **real time (RT)** considerations must be taken into account. It is often necessary to guarantee a maximum delay. In some situations it is unacceptable, if someone turns on the light and it takes a few hundred milliseconds or even seconds until the light really goes on for example. Compared to industrial automation, these real time requirements are not so hard. In industrial automation, timing constraints are often more important. Sometimes, events and results must be produced in a specified interval. In the industrial domain, it is often necessary

to meet hard deadlines[1]. Otherwise, a catastrophe could be the result. In building automation systems, these real time requirements are more moderate. It is important to note that they can still exist and so they must be considered. Therefore, common network protocols which are based on CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocols are often not practical.

As already mentioned, another important requirement is **fault tolerance** and **safety**. No system is perfect so it is important to protect the system against faults and their consequences. Therefore, it is essential to detect the faults in an early state. If the avoidance of such a fault is impossible, the consequences must be isolated. It is important that only a part of the system will be affected (a possible approach is described in Section 3.6). A single fault should not bring down the whole system. The affected part of the system should reach a fail-safe state where no critical results can happen. Additionally, it is sometimes necessary to guarantee a minimum of system performance. For example, if the lighting system fails, it is essential that the fire alarm system is still working. If a system provides such a minimum level of service in case of a failure, the system is called fail-operational. Especially, for critical systems these considerations are fundamental.

It is important to note that not only protection against faults caused by unpredictable system states is an important issue. Nowadays, **security** in building automation systems is getting more and more important. This means that protection mechanisms against attacks which are caused by humans must be implemented too. Better would be to design the system itself secure. As mentioned in Section 1, the importance of security was and is often played down and it is not considered with the appropriate importance as much as it should be. Therefore, this thesis will discuss this important topic extensively.

2.4.2 Network Services and Types of Communication

The network protocol must provide **basic network services**. First, it must be possible to add new devices easily. This means that integrating a new device in an existing network must be possible without too much configuration effort. For example, if the device connects to an existing network, a new network address must be assigned to this device. Additionally, the device must get enough information about the network and its topology (for example gateway addresses, QoS information, ...). Otherwise, the new device cannot talk to other network nodes. To provide this integration of new devices in an easy way, so called plug-and-play

[1]Deadline means the exact point in time at which a result must be produced.

features are beneficial (for example, see [44]).

Furthermore, the network must provide different types of communication services. First, it must be possible to exchange data between two network nodes directly. In building automation systems, it must be possible to configure and maintain the devices remotely. To achieve this, a so called management system is necessary. This management system provides services which are used to configure and change the behaviour of the devices. For example, with these management services the system integrator can change configuration parameters remotely. Furthermore, the management system often provides a service to upload a new user application to the different devices. For the rest of this thesis, this type of communication (often referred to as vertical communication) is called **management communication**. The corresponding network messages are called **management messages**.

To exchange management messages between two devices, a so called **point-to-point** connection is necessary. For example, a Client/Server connection is a typical point-to-point connection. Depending on the demands on the connection, connection-less or connection-oriented communication is necessary.

The values from the sensors itself must also be exchanged between the devices. This process data must be processed and transmitted to the actuators (horizontal communication). In this thesis, the transmission of process data is called **process data communication**. The corresponding network messages are called **process data messages**.

A building automation system often needs the possibility to send data values to a group of devices. For example, if a light switch controls more than one light, it must be possible to send the changing events to the corresponding device group efficiently. In this sense, efficiently means without too much communication effort. Therefore, a simple point-to-point connection to each device of the receiver group is not practical. A better approach is to use communication services like **broadcast and multicast**[2]. Compared to point-to-point communication, only a single message is needed to transmit the necessary data to more than one network node. Especially in building automation networks, this type of communication is very useful. In building automation systems, a publisher/subscriber model is often used for event notification. Many different types and variants exist. [24] describes these models in detail. This thesis will use a change-of-value (COV) model for further considerations. If a device wants to receive a particular event, this device

[2]Multicast is a special type of broadcast. Only the members of a logical group receive the messages with multicast .

subscribes to this event. If the event occurs (change-of-value), the responsible device (publisher) sends a multicast message to all subscribers.

Additionally, some process data may be exchanged directly between two devices (for example a manual override). For this type of process data communication, point-to-point connections are used.

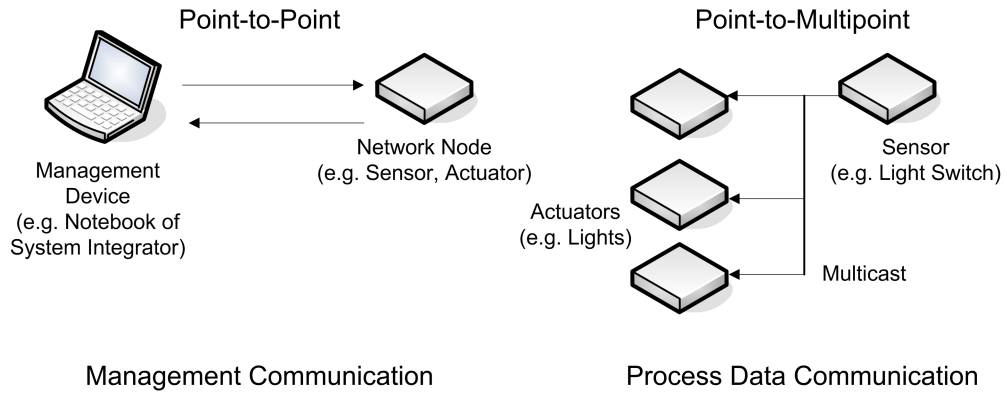


Figure 2.5: Management Communication vs. Process Data Communication

2.4.3 Control Network Media

In the field of building automation, different network media exist. It is important to distinguish between wired and wireless media. **Wired** media are best-known and mostly used. A wired medium uses physical cables as transmission channels. Generally, there are three important properties which influence each other.

First, different **network topologies** exist. The most common ones are ring, bus, star and free topology. Due to the required flexibility, many building automation solutions provide the opportunity to use free topology. With this topology, the electrical engineer is more flexible in placing the physical cables.

Another important property of the network medium is the **maximum cable length**. Normally, the length of one network segment is limited. If such a network segment is too small, repeaters can be used to extend the length of a network segment. These repeaters amplify the signal and so a larger transmission length can be achieved. It is important to note that normally a repeater does not decouple the network segments. So the maximum amount of repeaters is also limited.

The maximum **bandwidth** is also an important property. It varies depending on the maximum cable length, the topology and the type of the physical cable

itself from a few KBit/sec up to MBit/sec (A bandwidth of a few GBit/sec is possible but still uncommon and not necessary in the control level of building automation systems.). It is important to distinguish between the maximum theoretical bandwidth and maximum amount of user data which can be transmitted over the channel. Due to the protocol efficiency[3] and the byte coding, this maximum amount of user data is always lower than the theoretical bandwidth of the channel.

Another difference is the type of the physical cable itself. The most important cable types are twisted-pair (TP), coaxial, powerline and fibre optic.

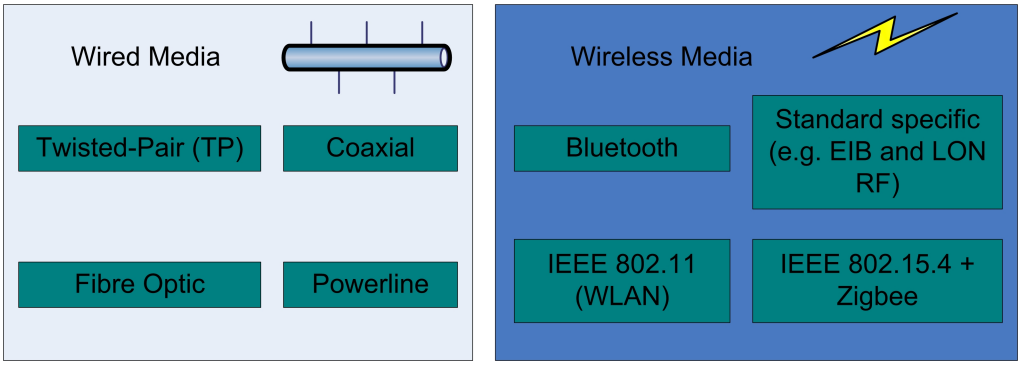


Figure 2.6: Different network media

Fibre optic is normally used for high-speed communication and therefore uncommon in the control level[4]. Therefore, fibre optic is rarely used (due to the cable costs, even at the backbone level twisted-pair is more common).

Every building has an electrical power supply network. Therefore, it is advantageous to use this existing network. If a **powerline** network is used as control network too, no additional network has to be installed. So installation costs can be saved. Unfortunately, a lot of problems have to be solved. First, these powerline networks are not designed to operate in high frequency range. A lot of other home appliances (for example microwave, washing machine and other white goods) are connected to this power supply network too. These devices generate impulses

[3]To manage the network communication, headers and trailers are added to the user data. These headers and trailers are unimportant for the application itself, but they consume network bandwidth too. Additionally, the protocol uses control messages which must be transmitted too (for example to establish a connection).

[4]LONWorks supports fibre optic.

and wideband noise which disturb the communication of the building automation system (an analysis of noise in powerline networks is given in [23]). Therefore, new modulation techniques have to be developed (for further details see [15] and [14]). Additionally, the powerline network must be separated from foreign powerline networks. The powerline network must be isolated to avoid an interference of other powerline networks. Eavesdropping must be prohibited too. Due to the radiation of the powerline medium, it is even possible to eavesdrop without physical access to the medium. Therefore, it is essential to encrypt the transmission channel (for further details, see Section 3.2).

Twisted-pair plays the most important role even in the field of building automation. Due to the decreasing costs of twisted-pair cables, it is more practical to install a separate network (especially if a new building is put up from scratch). Compared to powerline, twisted-pair is more reliable and robust. Therefore, twisted-pair is the most common medium. In earlier days **coaxial** cables were used. However, today coaxial cables are very uncommon and twisted-pair is used instead.

Wireless technologies are getting more popular recently. Due to further development, wireless technologies are flexible enough to become an alternative to hard-wired media. With a wireless communication approach, cabling and installation costs can be reduced. It is also possible to integrate moveable devices (for example a portable controller device or a position sensor mounted on a transport vehicle). To achieve this benefit, it must be possible to run the devices on batteries even for months or years. It is important to note that most wireless standards need a lot of power. Therefore, they are not practical for small embedded devices.

Due to that reason, the most common standard **IEEE 802.11** (known as WLAN) is not suitable for small embedded devices at the control level of building automation systems.

Another important short radio technology is Bluetooth. **Bluetooth** is a low cost and low power wireless technology which is designed for small devices. Bluetooth consumes less power than IEEE 802.11 (for example class 3 Bluetooth devices[5] need less than 1 mW).

Sometimes this power consumption is still too much for small embedded devices which run on batteries. Therefore, other technologies like **IEEE 802.15.4** (for further details see [17]) and **Zigbee**[6] are more practical.

[5]The Bluetooth specification 1.1 defines three different device classes. This classification depends on the maximum range and the power consumption of the device. For further details see [16].

[6]Zigbee is based on IEEE 802.15.4. Additional layers were added to provide additional features

In addition to these mentioned radio technologies, a lot of standards have their own wireless solution. For example, both LONWorks and KNX support wireless communication.

Since no physical cable is needed, safety and security issues must be carefully taken into account. In this field, a lot of development is done.

2.5 Device Classes

A building automation system consists of different network devices. As mentioned in Section 2.3, these devices are located at different network levels. At these levels, the devices must perform different tasks. Therefore, each network node must meet its individual requirements. Depending on these requirements, different types of building automation devices exist. Due to these types, the devices can be divided into different device classes. Based on the two-level model in Section 2.3, a classification into three different device classes is the most practical one for this thesis.

	Field Class	**Interconnection Class**	**Management Class**
Devices	Sensors, Actuators, Intelligent Devices	Gateways, Routers	Webserver, Logging Server, Workstation
Memory	low	middle	much
Computing Power	less	less	more
Environmental Factors	possible rough	normally mild	normally mild
Data Traffic	less	middle	middle or high
Communication Model	peer-to-peer	mostly peer-to-peer	mostly Client/Server

Table 2.1: Device Classification

Typical **field class** nodes are devices that are located at the control level. These devices interact directly with the physical environment. They are responsible for

like security and routing. For further details see [18].

data acquisition, data processing and for controlling the behaviour of the environment. To achieve this, different types of sensors and actuators are used. In earlier days, these devices were simple and therefore they provided only basic functionality. Today, the trend goes towards intelligent sensors and actuators. This means that such intelligent sensors and actuators provide additional functionality. To achieve this, microcontrollers are embedded in such devices that perform these additional tasks. For example, the network interface is often included in the embedded device. Additionally, a lot of configuration parameters can be changed to modify the behaviour of the device (for example changing the scaling of a sensor). Furthermore, different user programs can be uploaded into the memory of the embedded device. These user applications can be written in higher programming languages (for example in C). With this approach, it is possible to change the behaviour of the device. So, the integrator of the building automation system can choose which automation task the device should perform.

As mentioned in Section 2.3, these intelligent devices perform automation and control tasks too. Cost-effectiveness is essential in this class of devices. Since normally a lot of field class devices are used in building automation system, each device should be as cheap as possible. Additionally, due to the limited installation space the devices must be as small as possible. Therefore, most intelligent field devices contain small, low cost microcontrollers. This means that such embedded devices have limited memory resources and limited computing power. Furthermore, field devices should run stable for several years or even decades. This means that mechanical parts like cooling fans should be avoided. Therefore, only passive cooling systems are possible which means microcontrollers with limited computing power must be used[7].

Additionally, environmental factors like temperature and humidity must be taken into account. For example, outdoor sensors must be water resistant and insensitive against high temperatures[8]. Due to these reasons, the hardware of field class devices must be robust and resistant against these environmental factors. Therefore, they must be enclosed in a safe containment to avoid damage.

As mentioned in Section 2.3, at the control level, the transmitted data packets are smaller than in higher levels. This means that the field class devices do not need handle high data volumes. Therefore, the memory and the input/output

[7]Microcontrollers with higher clock rates dissipate more heat. Therefore, the CPU must be cooled enough to avoid thermal damage.

[8]For example depending on the allowed ambient temperature three different IC classes exist: ”commercial” from 0°C to 70°C, ”industrial” from -40°C to 85°C and ”military” from -55°C to 125°C.

buffers can be smaller. Furthermore, it is not necessary to use 32 bit or even 64 bit microcontrollers. In most cases, a 16 bit architecture is still enough (sometimes even 8 bit architectures are used).

Normally, field class devices use a peer-to-peer model for communication. Compared to a Client/Server model, a peer-to-peer approach is more practical at the control level. The advantages of peer-to-peer communication are better fault tolerance, better distribution of the traffic, less latencies in control loops and loose coupling. Unfortunately, peer-to-peer communication is more complex and therefore these systems are harder to design. A detailed comparison of these models is given in [63].

The devices of the **interconnection class** are responsible for the network communication. These devices interconnect different networks and network segments. Additionally, they are responsible for the correct routing of the data packets. Typical devices are repeaters, routers, proxies, bridges and gateways. These interconnection devices operate at different layers. Depending on this layer, these devices have different names (repeater, bridge, router, gateway, proxy,...). For this thesis, a distinction between routers and gateways is enough.

Depending on the networks that the device interconnects, two different types of devices exist. If a network device interconnects two different network types (here, different network types means different network protocols), the data packets must be translated (for further details see Section 2.3). If a device performs such a mapping, it is called a **gateway**.

If the protocols on both sides of the device are identical, a translation is not necessary. These devices are called **routers**. Routers are responsible for routing the network traffic. The data packets are only forwarded to the network segments in which the destination device is located. With this approach, the network traffic is decreased. To extend the maximum network segment length, routers reconstruct the physical signal too.

Like field class nodes, an interconnection device normally contains low cost hardware components. Compared to workstations, they have a low clock rate. As routing is not a very complex task, this limited computing power is still enough. The memory requirements depend on the task the interconnection device has to perform. It is common to add additionally functionality to this type of devices. Since these devices are located at topological important points in the network, they are ideal for performing logging or monitoring tasks. Therefore, interconnection devices often need more memory than field class devices.

Compared to field class devices, interconnection devices are normally located at central points in the building (for example in a switch cabinet or in a 19” rack).

This means that the environmental factors can be seen more relaxed. It is still important that these devices are robust and maintenance-free. So, it is recommended to embed the device in a safe containment (for example to protect it against dust).

Due to the fact that interconnection devices are responsible for interconnecting different networks, the data traffic is a little bit higher than in field class devices. To avoid performance bottlenecks, it is necessary to design the network carefully. Especially, the distribution of the interconnection devices is essential because it influences the network traffic significantly.

As mentioned in Section 2.4, an interconnection device can act as a server and a client (bottom-up and top-down communication). So compared to the field class devices, the communication model is not a strict peer-to-peer one. At the backbone side, the management nodes communicate through point-to-point connections with the gateways (Client/Server model). For process data communication a peer-to-peer model is used. To avoid bottlenecks and introducing single points of failure, more or less self-contained network segments are interconnected in a hierarchical manner.

The **management class** consists of devices that are necessary to configure and maintain the building automation network. They perform network management tasks like configuring network nodes or logging and storage of statistical data. Additionally, interconnections to other networks are possible (for example office networks, Intranet, Internet,...).

These management devices are normally located at the backbone. At this level, they have a global view of the whole network. It is important to note that it is also possible to integrate management devices at the control level. For example, an intrusion detection system can be installed at the control level to detect abnormal network traffic in a particular network segment (for further details see Section 3.7).

Management devices can be classified into two subclasses. Most of the devices are located at fix points in the network. These devices are **stationary**. Typical stationary devices are server stations (webserver, logging server, database server) and workstations (normal PCs, Terminals).

On the other hand, there are special devices that have not a fixed location in the network. Additionally, they are not always connected to the building automation system. This means that such a device joins the network, performs a particular task and disconnects after it has finished its job. In this thesis, such devices are called **temporary devices**. For example, a notebook or the PDA of a system operator can be such a temporary device. The system operator connects with his temporary device (wireless or at special access points) and performs configuration

or maintenance tasks. When he has finished his work, he closes the connection and the device does not belong to the network anymore. To achieve this, the network management must provide services which make an integration of temporary devices easy. Again these mechanisms can be used to gain unauthorised access to the network. Hence, it is important to protect these mechanisms carefully.

Compared to the number of field devices and gateways, the amount of management devices is much smaller. Therefore, the device costs are not so important. So, devices with a better performance can be used. This means that management devices normally have more computing power and more memory. If such a device is used to configure or monitor the system by humans, normal PCs are used.

Management devices are located in a relaxed environment. Server stations are normally integrated in special containments (switch cabinet or 19" rack). Management workstations are located in an environment which is similar to an office. Therefore, no extra considerations must be taken into account. Like all other devices in a building automation system, management devices should be long-lived. As normal PCs are often used as management devices, these devices can replaced more easily in case of a failure.

As mentioned before, a management device usually uses point-to-point connections. The management device establishes a connection to the particular device and performs configuration and maintenance tasks. So, these management devices normally use a Client/Server model for communication (management connection; see Section 2.4.2).

The amount of network traffic depends on the tasks that the devices have to perform. For example, in big building automation systems collection of detailed statistical data can produce a lot of network traffic. Therefore, management devices are often located at the backbone level where a higher network bandwidth is available.

Chapter 3

Security Theory and Concepts

As mentioned in Section 2.4, security is getting more and more important in building automation systems. Therefore, mechanisms must be implemented to protect building automation systems against security attacks. After a brief introduction into building automation systems, it is time to take a closer look now how these systems can be protected against security attacks.

Therefore, this chapter gives a general overview about basic security concepts and their importance for building automation systems. In this chapter different security mechanisms are going to be discussed. Additionally, it will be pointed out how these concepts can be implemented in existing building automation systems.

3.1 Introduction into Security

First, it is necessary to explain basic security definitions and terms. In the terminology of information technology, security means the protection of services against malicious attacks. To protect a system against such attacks, it is necessary to define a **security policy**. This policy is a formal specification that describes which actions are allowed and which actions are prohibited. Additionally, the policy must define which entities[1] are permitted to initiate certain actions.

To fulfil the security requirements which are defined in the security policy, different **security mechanisms** must be implemented. These security mechanisms protect the different services against security attacks.

[1] An entity can be a single person, a devices or a service.

To do a malicious attack, the attacker[2] must find a way to break into the system. Such a security attack is only possible, if a **vulnerability** exists. A vulnerability is the opportunity to cause damage ([19]). A weak cryptographic algorithm (for example an algorithm that uses short keys or that uses a simple mathematic function), a badly designed protocol (for example a network protocol which transmits cleartext passwords) or an insecure implementation (for example implementations with security holes like buffer overflows) can be vulnerabilities. To find a way to gain access to a system, the attacker must find such a vulnerability. After the attacker has found one, the attacker will try to exploit it. An **exploit** takes advantage of a specific vulnerability. Such an exploit is often a piece of software which provides the attacker the opportunity to gain unauthorised access to the system.

3.1.1 Security Threats

The author of [19] distinguishes between four different types of security threats. These four types are:

- Interception
- Interruption
- Modification
- Fabrication

Interception means an unauthorised entity has the possibility to access a service or data. For example, if an unauthorised person listens to the control network of a building automation system and tries to intercept data packets, this threat is classified as interception. Another typical example is when data is illegally copied (for example stealing the routing table of gateways to get more information about the network topology).

All kind of attacks with the objective of making a service or data unavailable are classified as **interruption**. Denial-of-Service attacks are typical interruption threats. For example, an attacker can try to flood a remote device with network packets. If the attacked device is not able to handle this amount of incoming packets, it will become unavailable (for example the internal message buffers overflow) . This class of attacks will be discussed in detail in Section 4.2.3.

A threat is classified as **modification**, if an unauthorised entity changes or tampers the behaviour of a service or the content of the data. In terms of building

[2] An attacker not only needs to be a human. In this thesis, an ”attacker” can also be a piece of a software or a host computer. Therefore, in the rest of this thesis, the neuter pronoun ”it” is going to be used.

automation, an example of modification would be where a sensor value is changed during transmission. Another example is where the application itself (the program code of the application) and therefore the behaviour of the device are modified.

Fabrication is referred to actions where additional data or an additional service is added which have not been specified. For example, the lighting system is modified in a way that an unauthorised person is able to switch off all lights in a building with a single action. Replaying of network messages is also a kind of fabrication. For example, if an attacker retransmits a previously sent ”open-door” message, this kind of attack is classified as fabrication.

3.1.2 Objectives of Security Mechanisms

As mentioned before, different security mechanisms are necessary to protect the building automation system against security threats. Depending on the type of protection, the security mechanisms must achieve different objectives. The author of [21] distinguishes between eight different security objectives:

- Confidentiality
- Integrity
- Availability
- Authentication
- Authorisation
- Auditability
- Non-repudiability
- Third-party protection

One of the most important objectives is **confidentiality**. This means that the disclosure of confidential information must be avoided. It must be guaranteed that only entities with the required privileges have access to confidential data. As mentioned before, these access rights must be defined in the security policy. For example, in an access control system, the confidential part of the user database (for example passwords)[3] must be protected against unauthorised disclosure. If the security mechanisms are using keys and passwords, these must be kept secret too.

Another important objective is **integrity**. If a security mechanism guarantees the integrity of data, the modification by unauthorised entities must be prohibited. For example, in a public backbone network of a building automation system, it

[3]This user database contains information about the user accounts like access rights, passwords or other authentication properties.

is necessary to avoid unauthorised modification of the data packets. Injection of new messages and replaying of old ones must be prohibited as well.

Even if the integrity and confidentiality of the transmitted data is guaranteed, the services and the data itself must be available. Therefore, the availability of the data is important, too. **Availability** means if services or process data are needed by an entity, the requested services and the corresponding data must be available whenever required. An attacker must not be able to deny the possibility to use services or data. To achieve this, an avoidance of denial-of-service attacks is necessary.

As mentioned before, it must be specified which entities are allowed to initiate actions and which entities are not allowed. To achieve this basic security concept, it is essential to prove the identity of the involved entities. Therefore, an **authentication** mechanism is necessary. For example, to guarantee in an access control system, that only an authorised entity is allowed to open a particular door, the identity of the initiator must be proven. After the identity has been successfully verified, it must be guaranteed that no other user is able to steal this identity. So, these identities and especially the currently valid ones must be managed in a secure manner. Authentication is one of the most important security mechanisms because it provides a base for a lot of other security mechanisms.

If the identity has been proven, it must be decided whether the entity has the necessary privileges to perform the requested operations. To achieve this, an **authorisation** mechanism is necessary. Authorisation is strongly related to authentication because without proving the identity, an authorisation does not make sense. Authorisation is also known as access control. It is important to note that authorisation is usually more than simple a ”access/deny” decision. Authorisation mechanisms often have to distinguish between different access levels. The entity needs to reach at least the access level that is required to perform the requested action. For example, in a building automation system only the system operator has the necessary privileges to change the full behaviour of a device (for example the right to upload a new user application). Other field devices are only allowed to change certain configuration parameters (for example the scaling of a sensor). To achieve this, the necessary information is often stored in an access control matrix which is saved in a database (for further details see Section 3.3.5).

Another important objective is **auditability**. To reconstruct the history of a system, it is necessary to store all relevant actions which are executed on the system. This approach of storing relevant actions is also referred to as logging. With this stored data, it is possible to detect security attacks in an earlier stage. For example to guess a password, an attacker tries many different passwords (brute force

attack). If such ominous actions are detected, further attacks can be avoided. A system that detects suspicious actions is called intrusion detection system (for further details see Section 3.7). The system logs can also be used to discover system failures. Depending on the size of the system, storing all actions is impossible. Therefore, it is essential to store only the information which is security relevant. Furthermore, it must be guaranteed that the attacker cannot modify the system logs. If the attacker is able to remove the corresponding entries of the system logs, he can cover its tracks.

Non-repudiability means to prove the identity of the entity that initiates an action, even if the initiator does not cooperate. This means that an initiated action cannot be denied later by the entity which was involved. Such a proof is often regulated by law. For example, if a safety system is shutdown, the identity of the person who initiated the shutdown must be proven irrefutably.

Third party protection means to avoid damage of other, external systems by use of the own system. It must be prohibited that an attacker uses the resources of a foreign host to attack another third party system. For example, distributed Denial-of-Service attacks use the system resources of different systems to get enough bandwidth to shutdown high performance systems. Worms are other well-known attack methods that use foreign system resources to attack third party systems are worms. After the worm has infected a system, it uses the resources of the infected system to attack other third party systems.

3.2 Securing the Transmission Channel

A network offers the possibility to exchange data between different network nodes. The data packets are transmitted over the network through a transmission channel. To guarantee the integrity and confidentiality of these data packets, these transmission channels must be protected. A transmission channel which is protected against interception, modification and fabrication is called a **secure channel**. A secure channel does not necessarily protect against interruption. A protection against interruption (for example against Denial-of-Service attacks) is not a trivial task. It is often necessary to implement additional security mechanisms to avoid such attacks (for further details see Section 4.2.3).

3.2.1 Protection against Interception

To achieve a secure transmission, it is necessary to encrypt the data messages. To encrypt the data packets, cryptographic algorithms are necessary.

The main objective of a cryptographical method is to encrypt a message m which is transmitted from a sender S to a receiver R. The original form of the message is called plaintext P, whereas the encrypted form is called ciphertext C. To get C from P, a mathematic function called encryption method E is used. This method E generates together witch a key K (K_E denotes the encryption key) C from P, that means:

$$C = E(K_E, P)$$

To obtain P from C, a decryption method D is needed. This mathematic function D computes together with the decryption key K_D P from C, that means:

$$P = D(K_D, C)$$

Figure 3.1 summarises the encryption/decryption process.

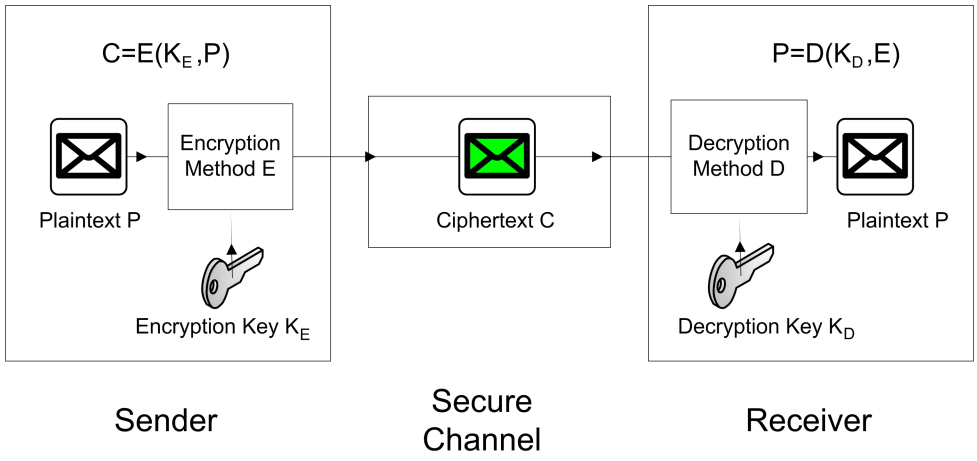


Figure 3.1: Encryption and Decryption of Messages

In the past, the algorithm itself was kept secret. This technique is very insecure. If the algorithm becomes public, the cryptographic method is not secure anymore. Therefore, modern cryptographic algorithms use another approach. These techniques keep the key secret, while the algorithm itself is made public. It is important that a cryptographic method becomes not weaker even if the source code of the algorithm is made public.

One of the most important characteristics is the **key length** of the algorithm. Attackers often try to guess valid keys. To achieve this, all possible keys must be tested whether they are valid or not (brute force attack). Therefore, it is often better to choose longer keys. If the key length increases, the amount of different keys also increases. This means that the attacker has to try more possibilities to find a valid one. A long key means the effort to encrypt and to decrypt messages increases. Especially in building automation system, this could cause problems. As mentioned in Section 2.5, field class devices have little computing power and therefore cryptographic methods that use long keys cannot run on small embedded devices.

In addition to that, the key length is often limited by the algorithm. A lot of algorithms were developed in a time where the performance of the available computers was too small to guess passwords in a reasonable time. Due to the increasing computing power, these brute force attacks need less time. Therefore, it is possible that a cryptographic algorithm becomes weaker the longer it is used.

Depending on the used encryption and decryption keys, two different types of cryptographic systems exist. If the encryption key and the decryption key are identical ($K_E = K_D = K$), the cryptographic algorithm is called **symmetric**. This means that

$$C = E(K, P), P = D(K, C)$$

Both S and R must know this key K. K must be exchanged in a secure manner and must be kept secret. Depending on the input data, two different types of symmetric algorithm exist. **Stream ciphers** process the input data bit- or byte-wise, whereas **block ciphers** encrypt the input block-wise.

Three examples of important symmetric algorithms are:

- RC4
- DES
- AES

Rivest Cipher 4 (RC4) is a byte-wise algorithm (for further details see [64]). Compared to other symmetric systems, RC4 needs limited computing power and limited memory. Because of that, it can be used in 8 bit microcontrollers.

The **Data Encryption Standard (DES)** is a block cipher (for further details see [31]). It has been used as a standard encryption algorithm for years. Due to the short key length (56 Bit), nowadays it is insecure for many applications. Therefore, it should not be used in new applications. However, to make DES stronger, the algorithm can be applied three times in a row. This form is also known as Triple DES.

Due to the weakness of DES, the **Advanced Encryption Standard (AES)** was intended to replace DES and Triple DES (for further details see [30]). As mentioned before, Triple DES is stronger than DES. Compared to DES, Triple DES is very slow in software and therefore impractical for small devices. As shown in Appendix A, AES is much faster (both in hardware and software) than DES and at least as secure as Triple DES. As it was developed by Joan Daemen and Vincent Rijmen, it is also known as the Rijndael cipher. It is a block cipher and its key size is 128, 192 or 256 Bits.

While symmetric systems use the same key for encryption and decryption, **asymmetric** algorithms use separate ones. This means that each entity needs a key pair K_E, K_D to encrypt and decrypt messages. In other words,

$$C = E(K_E, P), P = D(K_D, C)$$

Depending on the usage of the algorithm, one key is kept private whereas the other one is made public. Therefore, asymmetric algorithms are also referred to as **public-key systems**. For the rest of this thesis, K_A^+ shall denote the public key of entity A and K_A^- shall denote the private key.

Three well-known public-key systems are:

- RSA
- ElGamal
- Elliptic curve cryptography (ECC)

The security of **RSA** (named after the developers Ron **R**ivest, Adi **S**hamir and Len **A**dleman) is based on the fact that it is very difficult to factorise two large numbers into their prime factors (for further details see [29]). RSA is used in many cryptographic systems (for example in GnuPG; see [25]).

In the year 1985 Taher Elgamal developed the so called **ElGamal** algorithm (for further details see [28]). It uses the fact that it is very difficult to calculate discrete logarithms over a finite field. Since 1997 it is freely available and as a result of this, it has been used in many applications (for example in GnuPG; see [25]).

Elliptic curve cryptography (ECC) is based on the fact that it is very difficult to solve the discrete logarithm problem for the group of an elliptic curve over finite fields. ECC methods are at least as secure as RSA. Compared to RSA, ECC algorithms use smaller keys and therefore they are faster. For further details see [26].

Symmetric systems are much faster than asymmetric ones. Therefore, they can run on small embedded devices which have limited computing power. For example, RC4 can even be used on 8 bit microcontrollers. Additionally, symmetric

algorithms need limited memory than asymmetric ones. As mentioned in Section 2.5, field class devices have limited hardware resource and therefore symmetric algorithms are more appropriate in the building automation domain.

As mentioned in Section 2.4.2, in building automation systems group communication models like multicast and broadcast are often used to exchange process data. If this communication in a group of members must be protected against interception and modification, the process data communication must be encrypted.

If a symmetric algorithm is used, there are two possibilities. The simplest way is to share one key between all group members. As soon as one member is compromised, the key is not secure anymore and hence a new key must be generated. Especially, if a group has a lot of members, the risk that the key gets compromised increases.

Another solution is to use a separate key for every possible transmission channel. This means that each pair of group members have their own secret key. If one key becomes public, only one pair of members must generate a new key, while the other members can keep their shared keys. In a network group with n nodes, $n(n-1)/2$ keys are necessary with this approach. Hence an implementation of this scheme is nearly impossible, if a communication group has a lot of members.

It is important to take the relationship between the group members into account. For example, if a publisher/subscriber model is used, communication is normally performed between the publisher and the different subscribers. This means that it is not necessary that every possible pair of group members must have its own key. Since two subscribers that only receive change-of-value events do not need to communicate with each other, they do not need to share a secret key. For example, in a lighting system a light switch does not normally communicate with another light switch. Therefore, shared keys are only necessary between the switches and the light itself.

Another problem is the key exchange itself. It must be guaranteed that every member of the communication group gets the key in a secure manner. Again, if the group contains a lot of members, such a key exchange can be very complicated. If an asymmetric algorithm is used, every member of the communication group needs one key pair. This means that in a group of N members, every member needs its own public/private key pair. A problem with this approach is the distribution of the public keys. If A wants to communicate with B, B must know the public key K_A^+ of A and vice versa. Therefore, a so called key management facility is necessary (for further details see Section 3.3). As mentioned before, asymmetric algorithms are slower than symmetric ones. Therefore, it is often impossible to run asymmetric cryptographic algorithms on embedded devices.

To combine the benefits of symmetric and asymmetric cryptographic systems, a **hybrid** solution is often used. This means that one communication participant generates a secret session key. To exchange this session key, an asymmetric algorithm is used. After this secret session key has been exchanged in a secure manner, a faster symmetric method can be used for further data transmission. This scheme has another important advantage. If the connection and therefore the session is closed, the session keys becomes invalid. This means that if a new session is started, the old key which was stolen in the session before is not valid anymore. Therefore, if an attacker gets the session key, only a single session is affected. Generally, the time interval where a secret key is valid should be limited.

An example of a public-key solution is the well-known **Diffie-Hellman key exchange** (see [32]). With this scheme it is possible to exchange a shared key across an insecure channel. This key exchange protocol works as follows:

Entity A and B want to establish a secure connection across an insecure channel. First, they agree on two numbers g and p where g is a fixed primitive element of a finite field $GF(p)$ (for example $p = 23$ and $g = 3$). This two numbers can be made public. Then A chooses a secret number a (for example $a = 6$) and afterwards A sends $g^a \mod p$ to B (in our example $3^6 \mod 23 = 16$). B chooses also a secret number b (for example $b = 15$) and sends $g^b \mod p$ to A (this means $3^{15} \mod 23 = 12$). Now, A calculates $(g^b \mod p)^a \mod p = g^{ba} \mod p$ (this means $12^6 \mod 23 = 9$) and B calculates $(g^a \mod p)^b \mod p = g^{ab} \mod p$ (this means $16^{15} \mod 23 = 9$) . Due to the fact, that $g^{ba} \mod p = g^{ab} \mod p$ is true, both sides have now a shared key (in this example 9). It is important to realise that the private numbers a and b have never been transmitted across the network. An illustration of this key exchange algorithm is shown in Figure 3.2.

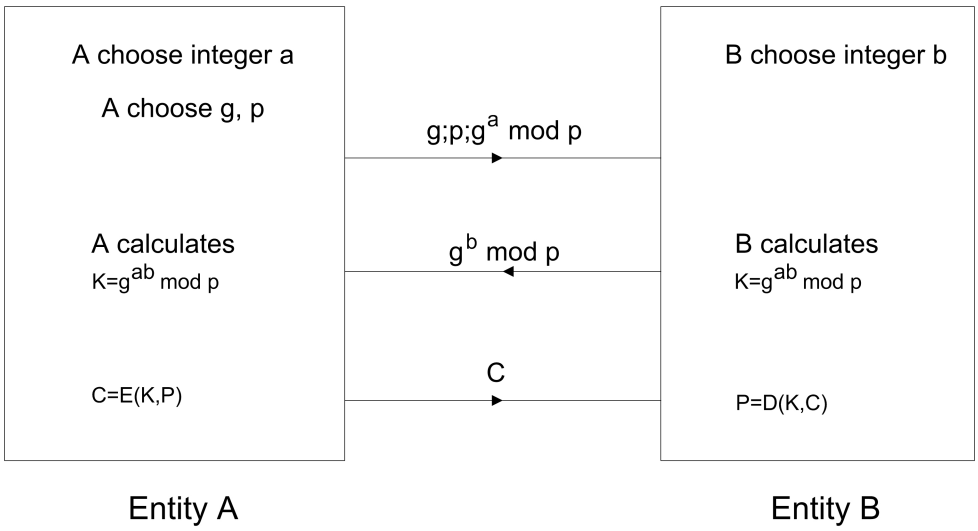


Figure 3.2: Diffie-Hellman key exchange

The Diffie-Hellman key exchange is based on the fact that it is very difficult to calculate logarithms over a finite field $GF(p)$ with a prime element p. This means that it is very easy to calculate $y = g^x \mod p$, but it is very difficult to calculate $x = \log_g y \mod p$. As p, g can be made public and a, b must be kept secret, the Diffie-Hellman key exchange is regarded as a public-key system.

3.2.2 Protection against Fabrication

An encryption protects the transmission against unwanted interception. It is important to note that it is still possible to replay messages even if the attacker does not know the plain version of the message (replay attack). For example, in a building automation system an attacker can simply replay an encrypted "Open door" message which was sent previously. If the receiver does not discover such a replayed message, the receiver will perform the requested action. To do such a replay attack, the attacker must get enough information about the network traffic. If the intruder listens to the network long enough, it is possible to guess the meaning of different messages. Or the attacker can simply replay different messages and observe what consequences these replayed messages have.

To avoid such a fabrication, messages often contain a so called **nonce**. A nonce is a number which is used only once. This nonce uniquely identifies the message.

Due to this nonce, two messages which contain the same user data (for example "open door") are not identical. To achieve this requirement, it is necessary that the set from which these numbers are chosen is large enough. To have a "perfectly secure" nonce, the set must be large enough to guarantee that each nonce is unique during the whole life cycle of the system. To avoid interception of this nonce, it is encrypted, too.

To achieve this protection against replay attacks, a counter is often used as a nonce. This counter is incremented after each transmission. If the counter is large enough, two messages with the same user data will always be different. So, if an attacker intercepts a previously sent message and replies it later, the receiver will be able to detect this replayed message. It is important to keep the counters on both sides synchronised. Otherwise, a correct message would be identified as an invalid one. A problem with this approach is the loss of messages. If a message gets lost and if the sender does not recognise it, the counters will get asynchronous. A possible solution is to accept counter values in a specified interval $(N, N + e)$, where N denotes the current counter value and e is the amount of messages, which can get lost.

Figure 3.3 illustrates an encrypted communication between two entities that use a counter to protect the transmission against replay attacks. Both the sender S and the receiver R initialise their counters with the same value N_1. Then S sends an encrypted message which contains the message itself and the counter N_1. R receives the message and decrypts it. After it has been decrypted, R compares the received counter values with the current valid one. If they are identical, the message is a valid one. Now, both S and R increase their counters.

During the transmission of this message, an attacker intercepts it. Later on, the attacker replies this message. Again, R receives the message and decrypts it. Now, the received counter and the current valid one are not identical. And so, R has detected this replayed message and R will discard the message.

Finally, S sends a new message. B receives the message and compares the received counter with the reference one. If they match, B can be sure that this message is not a replayed one.

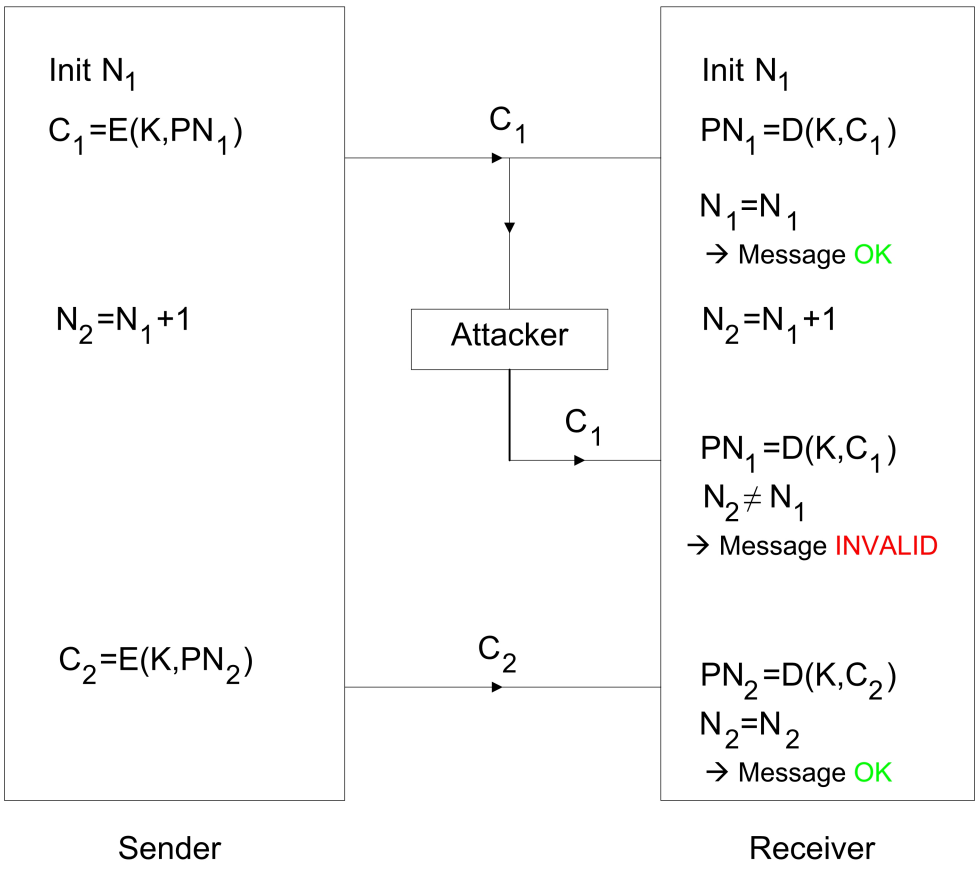


Figure 3.3: Protection against Fabrication

3.2.3 Protection against Modification

If the data packets are encrypted and if a nonce is added to the user data, the transmission is protected against interception and fabrication. Anyhow, there are still possibilities to disturb the communication between two entities.

As mentioned before, a nonce helps to discover replayed messages. With this approach, the receiver can detect a previously sent messages and so it can discard the replayed message. Unfortunately, it is still possible to modify the message. If an entity receives such a modified message, this message can be misinterpreted and hence it could cause damage. Even if the attacker does not exactly know the consequences of the modification, he can try to send many different modi-

fied messages and hope that such a message disturbs the normal operation of the receiver.

To protect the data packets against unwanted modification, a digest can be used. A message digest h (also called digital fingerprint) is generated by a one-way hash function H. In other words ($l(x)$ denotes the length of the binary string x),

$$h = H(m), \forall m : l(h) = const$$

A cryptographically secure hash function must have the following properties (see [20] and [21]):

- It is computationally infeasible to find m from h. On the other hand, it must be very easy to calculate h from m.
- The output h must have a fixed length for all different inputs m (even if the length of m is different).
- A modification of only one bit of m must lead to a modification of about half of the bits of h.
- If an input m and the corresponding output h is given, it must be computationally infeasible to find another input m' such that $h = H(m) = H(m')$.
- If only the hash function H is given, it must be computationally infeasible to find two inputs m and m' such that $H(m) = H(m')$.

Cyclic redundancy check (CRC) algorithms are used to calculate a checksum. This checksum can be used to detect transmission failures. Due to the linear structure of the used CRC polynomials, it is easy to change the input m without changing the checksum h. Therefore, the checksum must be encrypted too or a strong cryptographic hash function should be used instead.

The most widely used cryptographic hash functions are **MD5**(for further details see [33]) and **SHA**. Since many collisions were found in MD5, it should not be used anymore. It is recommended to use a SHA instead. Depending on the length of the output, the Secure Hash Standard (SHS) defines different types of SHA algorithms (SHA-0, SHA-1 and SHA-2). Generally, it is more secure to use a higher number of hash bits (for further details see [34]).

To protect the transmission of messages against unwanted modification, one of these hash functions can be used. To achieve this, the sender calculates the message digest of the message. To protected this calculated digest against modification, the sender encrypts it. Otherwise, an attacker could modify the message and calculate a new message digest.

After the sender has encrypted the message digest, this encrypted digest is sent together with the user data to the receiver. It is important to note that the user

data itself does not need to be encrypted. To detect an unauthorised modification, only the digest has to be encrypted. After the receiver has received the message it decrypts the received message digest. Additionally, it calculates the digest of the received user data. If the calculated and the received digest are identical, the receiver can be sure that nobody has modified the message.

Figure 3.4 illustrates this concept.

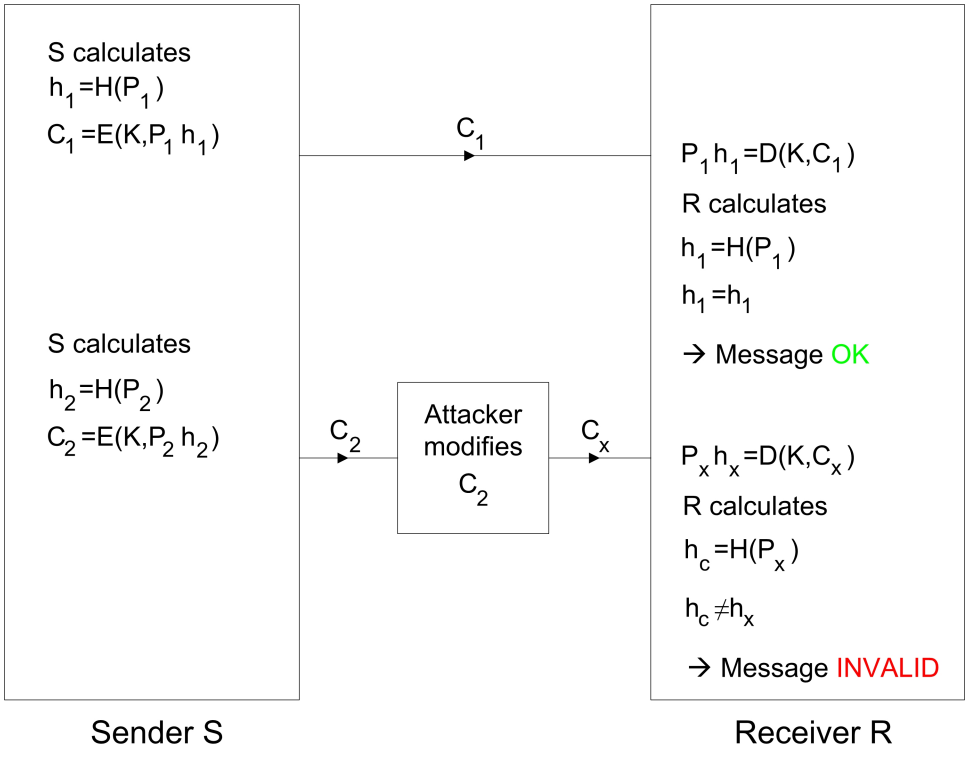


Figure 3.4: Protection against Modification

There are two ways, how this encryption can be achieved. First, it is possible to use a symmetric encryption. If the message digest is encrypted using a shared secret key, the receiver can be sure that only an owner of this secret key is able to calculate the hash value. Without knowing this secret key, an attacker is not able to calculate a new message digest.

Another possibility is to use a public key system. This scheme works as follows. The sender of a message calculates the hash value of the message. To protect this digest against modification, the sender encrypts it with its private key.

When the message arrives at the receiver, the receiver can decrypt the digest by using the public key of the sender. As long as the validity of the public key can be guaranteed, the receiver can be sure that nobody has modified the digest. As the digest is uniquely tied to the private key, this encryption digitally signs the message. Therefore, such an encrypted message digest is also called a **digital signature**.

3.3 Authentication and Key Management

In the previous section, methods and concepts were presented which guarantee the integrity and confidentiality of the transmitted data. With these methods, the communication participants can be sure that the data packets are transmitted over a secure channel. It is important to note that there is still a problem that remains unsolved. Even, if message integrity and confidentiality are guaranteed, it must be proven that the participants are what they claim to be. To achieve this, the identity of both communication participants must be verified. For example, in an access control system it must be guaranteed that only an authorised entity is allowed to open a door. Therefore, the sender of an ”open door” message must prove its identity.

It is obvious that authentication and message integrity/confidentiality is closely related. In many cases, authentication makes only sense, if the message integrity and confidentiality can be guaranteed. For example, if the communication protocol of an access control system supports authentication, the sender of an ”open door” message can prove its identity. If the integrity of the message cannot be ensured, it could be possible that an attacker has modified the received message.

It is important to note that a communication protocol that guarantees message integrity without implemented authentication mechanism does not make sense. As shown in the above example, it is essential to verify the identity of the communication participants. Otherwise, it cannot be ensured whether the sender of an ”open door” message has the necessary access rights to open the door.

There are several ways to verify the identity of a communication participant. These different of authentication mechanisms are explained in the next sections of this chapter. Based on [35] and [20], this thesis distinguishes between four different types of authentication protocols.

3.3.1 Authentication based on secret tokens

First, a so called secret token can be used to prove the identity of a communication participant. This secret token is sent to the entity which wants to verify the identity (see Figure 3.5).

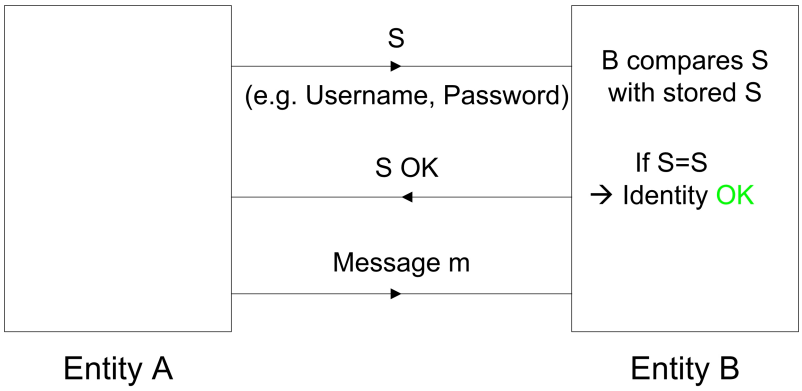


Figure 3.5: Authentication based on a secret token

For example, a PIN code, a username/password pair or a passphrase can be used as secret tokens. It is quite obvious that these secret tokens must be protected against interception. If the secret token is not transmitted over a secure channel, an attacker can simply intercept the secret token.

3.3.2 Authentication based on secret keys

As mentioned before, the use of a secret token which is transmitted in cleartext, is insecure and should be avoided. A more secure approach is to encrypt all messages that have to be sent during the authentication phase. If a symmetric algorithm is used, the shared secret key K can be used to prove the identity of the communication participants. This means that if an entity receives a message which is encrypted with a shared secret key K, the receiver of the message can be sure that only an owner of the key K can encrypt the content of this message.

The main disadvantage is that this method only proves that an owner of the key has sent the encrypted messages. This means that if an attacker steals the secret key, the attacker is able to authenticate successfully. As long as only the communication participants know the secret key, it can be verified whether the participants are what they claim to be.

As mentioned in the Section 3.2, an encryption only protects against unwanted interception. Therefore, a nonce is often used to avoid replay attacks. Figure 3.6 shows a possible authentication protocol which is based on a shared secret key.

First, the entity A sends an identity token A to B. This identity token is only used to inform B that A wants to prove its identity. This token can be transmitted in cleartext since it does not influence the security behaviour of the protocol. After B receives this token, it generates a nonce N_1 which is sent to A. A encrypts the received nonce N_1 with the shared secret key K and sends the encrypted version of N_1 back to B. To verify the identity of A, B decrypts the received nonce and compares it with the original one. As long as only A and B know the secret key K, B can be sure that A has sent the message. Additionally, if A wants to verify the identity of B, A generates also a nonce N_2 and transmits it to B. After B has received the nonce N_2, B encrypts it using the secret key K. Now, B sends this encrypted version back to A. After A has received the encrypted version of N_2, A can also verify the identity of B by decrypting the nonce. To achieve this, A decrypts the nonce and then compares it with the original one.

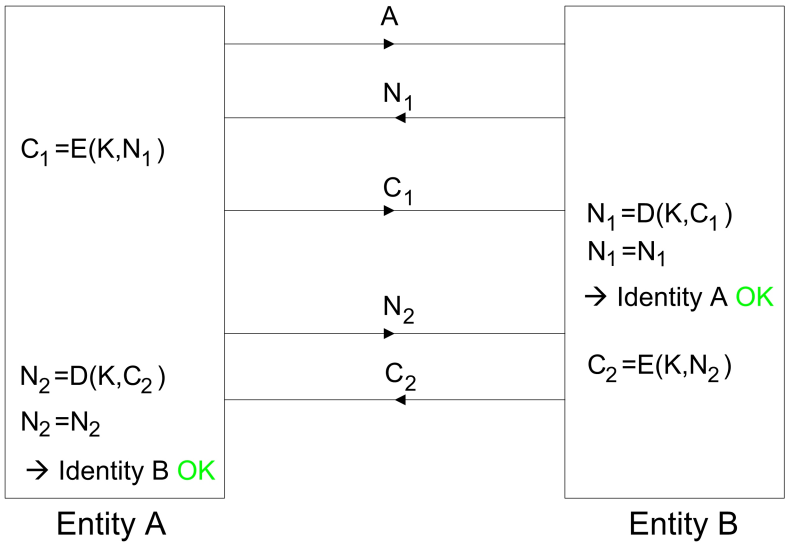


Figure 3.6: Authentication based on a shared secret key

Designing a secure protocol is not always a trivial task. [20] illustrates this by giving an example how an optimisation of an authentication protocol can make it vulnerable to security attacks.

Consider the following example which is discussed in [20]. To decrease the amount of necessary messages which have to be sent, the authentication protocol mentioned above (Figure 3.6) can be optimised. For example, entity A can send its identity token A and its generated nonce N_2 together in one message. Additionally, entity B can send its response C_1 together with the nonce N_1. As shown in Figure 3.7, this optimisation would reduce the amount of necessary messages to three.

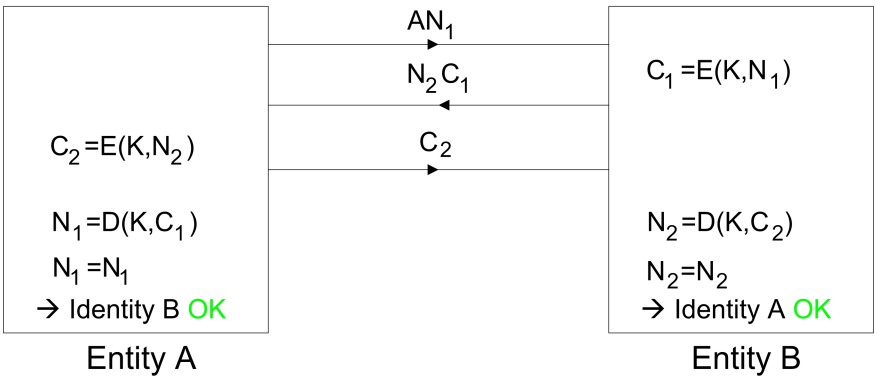


Figure 3.7: Optimisation of the Authentication Protocol in figure 3.6

This version of the protocol has a potential vulnerability (Figure 3.8). Considering the situation, where an attacker X wants to fake the identity of A without having the secret key of A. Suppose that X wants to authenticate to an entity B. X sends the identity token A together with a generated nonce N_1. B responds with another nonce N_2 and the encrypted version of N_1. To verify the identity of A, B waits until X responds with the encrypted version of N_2. As X does not have the shared secret key K, X cannot encrypt N_2 and therefore it cannot finish the authentication process.

To fool B, the attacker starts a second session. It sends the identity token A together with the nonce N_2, which it has to encrypt to finish the first session. B encrypts N_2 and sends it back to X. Now, X has the encrypted version of N_2 and so X can finish the first session by sending the encrypted version of N_2 to B. Since the response from the second session is reflected, this type of attack is also called reflection attack.

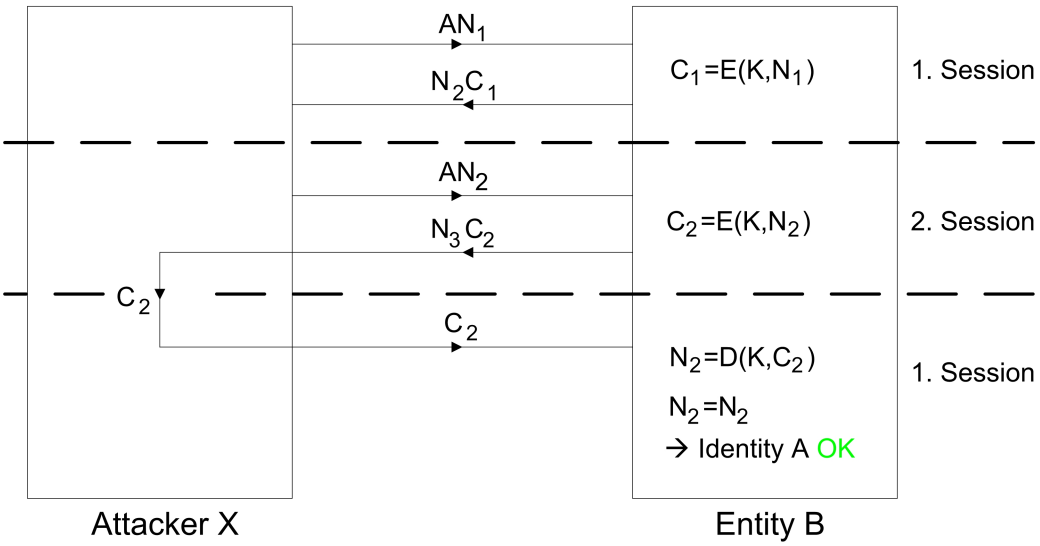


Figure 3.8: Reflection attack

This example shows how difficult it is to optimise a security protocol. As mentioned in Section 2.5, especially the control level of a building automation system contains small embedded devices which have limited hardware resources. To save memory and to decrease the execution time, it is often necessary to optimise the software implementations. It is essential that an optimisation will not impact the security of the protocol(for further detail see 3.4).

3.3.3 Authentication using a third party

Authentication methods which are based on shared keys have one disadvantage. These methods use symmetric algorithms to prove the identity of the communication participants. As mentioned in Section 3.2.1, there are two possibilities how a group of network nodes can use a symmetric algorithm to encrypt the transmission.

First, all members of a group can share one secret key. If an authentication protocol uses this approach, the identity of a communication participant cannot be clearly verified. This means that if a member of a group receives a message that is encrypted with a shared group key, the receiver knows that only an owner of the key is able to encrypt this message. As all members of the communication group know this key, it cannot be exactly determined which group member has

encrypted the message. The receiver can only be sure that a group member has sent this message (of course if nobody has stolen the key). Therefore, this scheme is not suitable for proving the identity of a single network node.

Another possibility is to use a separate key between each pair of group members. This means that if a group has N members, it is necessary to have $N(N-1)/2$ keys. It is obvious that for large N, this approach is impossible. Therefore, an alternative has to be found.

One possible solution is to use a trusted third party which manages the secret keys. Since this third party manages the different keys, it is often called **Key Distribution Center (KDC)**. This KDC is responsible for the distribution of session keys. If an entity wants to set up a connection with another entity, it must retrieve a session key. To achieve this, the entity sends a request to the corresponding KDC. The KDC generates a session key and sends this session key to both communication participants. With this approach, each entity only needs one shared key. This key is used to establish a secure connection to the KDC. It is not necessary to share a secret key with another entity.

Figure 3.9 illustrates this scheme. Suppose, entity A wants to set up a connection to entity B. First, A sends the identity tokens A and B to the KDC. The KDC receives them and generates a session key $K_{A,B}$ which can be used by A and B to set up a secure channel. After the session key has been generated, the KDC encrypts this session key using $K_{A,S}$ (C_1). Additionally, the KDC encrypts the session key with $K_{B,S}$ to get C_2. Now, these two encrypted versions of $K_{A,B}$ are transmitted to A. To get the session key $K_{A,B}$, A decrypts C_1 using its shared key $K_{A,S}$. Additionally, A sends C_2 to B. As C_2 offers B the possibility to communicate with A, C_2 is also called a ticket. After B has received the ticket, B can decrypt C_2 with $K_{B,S}$. Now, A and B both have the same secret key and so they can communicate through a secure channel. As long as the secret keys $K_{A,S}$ and $K_{B,S}$ are kept secret, only A, B and the KDC know the session key. Therefore, the identity of A and B can be verified.

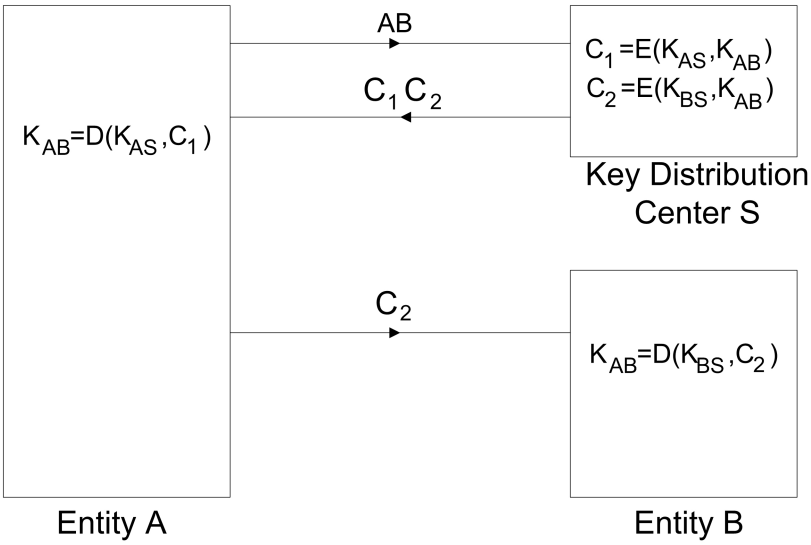


Figure 3.9: Authentication using a KDC

The protocol mentioned above should only explain the basic concept of a protocol that uses a KDC. It is obvious that there are a lot of problems which have to be considered. For example, there is no protection against replay attacks.

A more secure example of an authentication protocol which uses a central key server is called Needham-Schroeder protocol. It is named after its developers Needham and Schroeder (for further details see [36]). It has a lot of mechanisms implemented which make the authentication process more secure. For example, it uses a nonce to avoid replay attacks. A mutation of this protocol is used in Kerberos.

3.3.4 Authentication based on a public key system

Another possibility is to use a public key system for authentication. As mentioned in Section 3.2.1, in a public key system every entity has its own public/private key pair. These authentication mechanisms are based on the fact that a message which is encrypted using a public key can only be decrypted with the corresponding private key and vice versa.

Suppose that entity A wants to verify the identity of an entity B. To achieve this, A encrypts a nonce N using the public key K_B^+ of B and sends the encrypted nonce to B. B decrypts the received message with its private key K_B^- and sends

the nonce N back to A. Since only the private key K_B^- of B can decrypt a message that was encrypted with the public key K_B^+ of B, B has proven its identity. If B wants to verify the identity of A, B could achieve this exactly the same way.

Figure 3.10 illustrates this method. It is important to note that it is also possible to send the nonce in plaintext. If this approach is used, B encrypts the received nonce using its private key. Then it sends the encrypted nonce back to A. To verify the identity of B, A decrypts the nonce using the public key of B. If the nonce is valid, B has proven its identity.

It is important to note that these examples show only the basic concept of this scheme. There are still a lot of security problems which have to be considered.

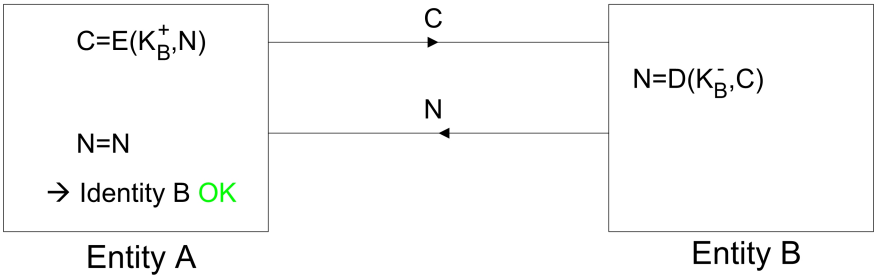


Figure 3.10: Authentication based on a public key system

As mentioned in Section 3.2.1, asymmetric algorithms are much slower than symmetric ones. Therefore, it is more common to use hybrid solutions. This means to exchange the session key and to verify the identity of the participants, an asymmetric algorithm is used. After the identities have been proven and after the session key has been exchanged, a symmetric algorithm is used for further communication.

Figure 3.11 shows, how such a hybrid protocol can be realised. This solution works as follows. Entity A generates a nonce N_1. This nonce is encrypted together with the identity token A using the public key K_B^+ of B. Then the encrypted message C_1 is sent to entity B. B receives the message and decrypts it with the corresponding private key K_B^-. Now, B generates a nonce N_2 and a session key K. At that point, B encrypts the decrypted nonce N_1 together with the nonce N_2 and the session key K using the public key K_A^+ of A. This encrypted message C_2 is transmitted to A. To get N_1, N_2 and K out of the received message C_2, A decrypts C_2 using its private key K_A^-. Now, A is able to verify the identity of B. As long as the public key of B was the valid one and as long as only B knows the

private key K_B^-, A can be sure that B has sent the message. To finish the protocol, A encrypts the received nonce N_2 with the session key K and sends the encrypted version of N_2 to B. After B has received the message, B is able to verify the identity of A. At that point, both communication participants have proven their identity. For the rest of this session, they can use the shared session key K to encrypt further messages.

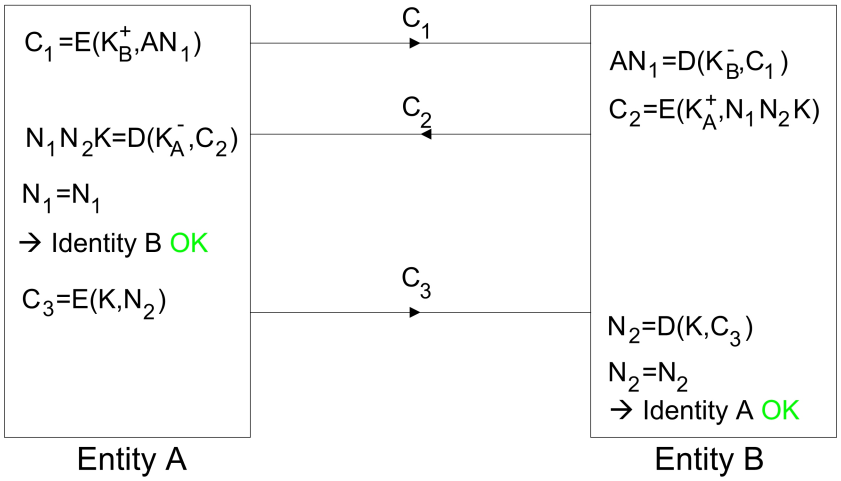


Figure 3.11: Authentication based on a hybrid system

As shown in the described example, an asymmetric algorithm can be used easily to prove the identity of the communication participants. Anyway, there are still problems which have to be considered.

First, this approach works only as long as the private keys of the entities are kept secret. If someone steals the private key of an entity, the identity cannot be proven anymore. Therefore, it is necessary to implement a mechanism which makes it possible to generate a new key pair. Additionally, if a new key pair is generated, the new public key must be distributed across the network.

Another problem is the distribution of the public keys. In a public key system it must be guaranteed that a public key belongs to the corresponding entity. A communication participant must be sure that it gets the correct public key.

To solve these problems, a key management facility is necessary. This facility has the objective to maintain and to distribute the public keys in a secure manner. It must also provide services which make it possible to revoke the validity of a public

key and to upload a new one. Section 3.3.6 shows how such a key management can be realised.

3.3.5 Authorisation

As mentioned at the beginning of this chapter, a security policy defines which entities are allowed to initiate certain actions. In the previous section, authentication techniques were discussed which make it possible to verify the identity of the communication participants. After the initiator of a request has proven its identity, it is necessary to check whether the entity has the necessary access rights to perform the requested action. For example, in a building automation system only the system operator is allowed to change configuration parameters of a field device. Therefore, it is necessary to verify whether the initiator of the request has the necessary access rights to change these parameters.

To achieve this, an authorisation[4] mechanism is necessary. This authorisation mechanism has the objective to decide whether an entity (called subject S) is allowed to perform an action a on a particular device (called object O). A common solution is to store these access rights in an **access control matrix**. A row of such a matrix represents a subject whereas a column represents an object. To verify whether a subject S is allowed to initiate an action a on an object O, this action a must be listed in the entry $M[s, o]$ of the corresponding access control matrix M. In other words, the matrix entry $M[s, o]$ lists all actions of object O which subject S is allowed to perform.

A building automation system normally has a lot of devices. If an access control matrix is used to store the necessary access rights, this matrix can become very large. Additionally, a device does not need the whole access control matrix. Depending on the task which a device has to perform, only a part of the matrix is relevant for the particular device. Therefore, it is more efficient to distribute the access control matrix.

If the access control matrix is distributed column-wise, each device gets is own **access control list**. This list defines which entity is allowed to perform which actions. With this approach, the devices need not store the whole access control matrix. Each device stores and maintains its associated access control list. If an entity wants to perform a particular action on a device, the device verifies the

[4]Some publications distinguish between access control(verifying access rights) and authorisation (granting access rights). These two terms are strongly related to each other. In this thesis, the term authorisation is used as a synonym for both .

identity of the initiator. After the initiator has proven its identity, the device checks the access control list whether the initiator has the necessary access rights. If the initiator is listed in the access control list and it has the necessary access rights, the action is executed. Figure 3.12 illustrates this approach.

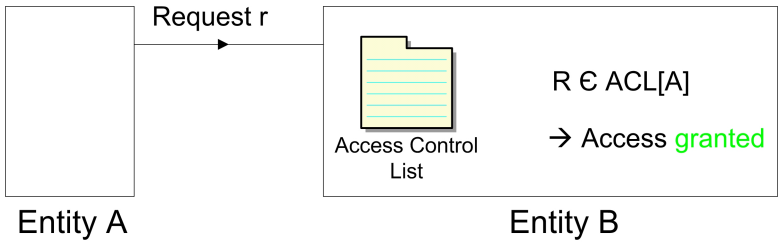


Figure 3.12: Access Control List

Another possibility is to distribute the access control matrix row-wise. This means that each subject has its own list of actions which it is allowed to perform on a particular device. The subject itself maintains this list of **capabilities**.

A capability is used like a ticket. It gives the owner the right to perform a certain action. If an entity wants to initiate an action, it sends the necessary capability together with the request to the particular device. The device receives the capability and verifies the authenticity of the received capability. If the capability is valid, the requested action is executed. It is obvious that this capability must be protected against unauthorised modification. Otherwise, an attacker could modify an existing capability and gain unauthorised access. Figure 3.13 illustrates this basic concept.

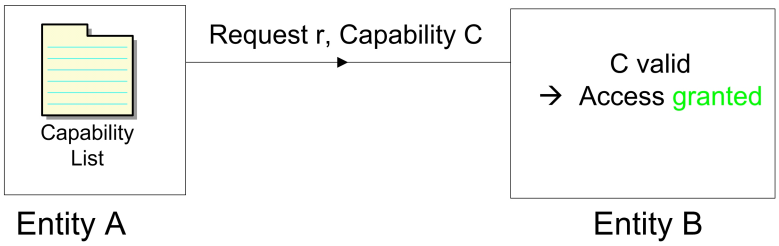


Figure 3.13: Capability List

3.3.6 Key Exchange and Key Management

Authentication protocols which use a third party or which are based on a public key system have one disadvantage. These protocols need a key management facility to manage the keys.

There are two ways to realise a key management. First, it is possible to use a centralised key server (**centralised approach**). This key server has the objective to maintain and to distribute the keys. Depending on the used cryptographic algorithms, two different types of keyservers exist.

As mentioned in Section 3.3.3, a **key distribution center (KDC)** manages the shared, secret keys which are needed by symmetric algorithms. If an entity wants to communicate with another entity, it sends a request to the KDC. The KDC generates a session key and sends it back to the entity. It is obvious that the communication between the entity and the KDC itself must be secured. Therefore, each communication participant needs a shared secret key which allows the entity to set up a secure channel to the KDC (for a detailed description see Figure 3.9 in Section 3.3).

If a public key system is used, it is not necessary to exchange a shared secret key between the involved communication participants. Unfortunately, there is still one problem which has to be solved. If an entity wants to encrypt a message with the public key of the receiver, the sender must be sure that the public key belongs to the entity that it purports to belong to. Again, a trusted third party can be used to solve this problem. If a public key system is used, this trusted third party is called **certification authority (CA)**. It has the objective to maintain and to distribute the public keys in a secure manner. If an entity is interested in a public key, it sends a request to its trusted CA. This CA sends a certificate back to the entity. This certificate contains the requested public key and the identity token of the owner of the public key. For example, in a building automation system the unique device number in the network could be used as identity token. To prove the authenticity of such a certificate, the whole certificate is digitally signed by the CA. As mentioned in Section 3.2.3, this means that a message digest is appended to the message. This message digest is encrypted with the private key of the CA. To verify the authenticity of the certificate, the entity tries to decrypt the message digest using the public key of the CA. Additionally, it calculates the message digest of the received certificate. If the decryption is successful and if the decrypted message digest and the calculated one are identical, the identity of the CA has been proven. Figure 3.14 illustrates this concept.

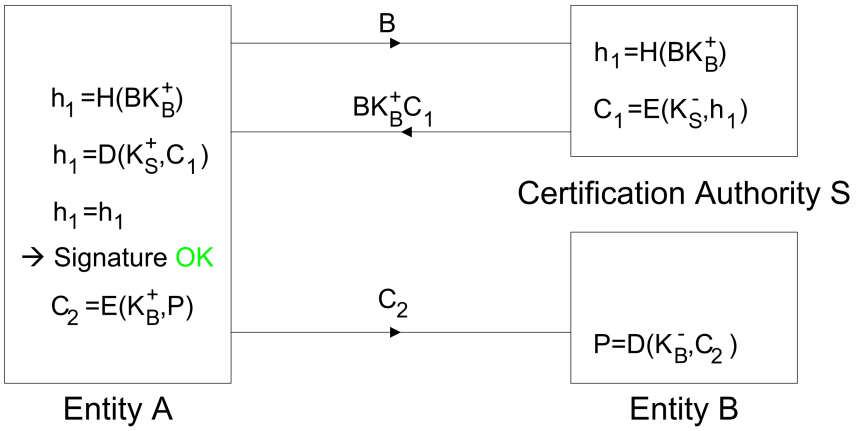


Figure 3.14: Certification Authority

It is important to note that each entity needs a set of trusted keys. If a KDC is used to distribute session keys, each entity must share a secret key with the KDC. Without this key, it is not possible to set up a secure channel to the KDC. If a public key system is used, each entity must trust the authenticity of the CA's public key. Without the public key of the CA, it is not possible to verify the authenticity of the certificate.

There are many different ways to solve this problem. One possible solution is to integrate the required keys into the client application itself. For example, a set of trusted public keys is integrated in the binaries of web browsers. In a building automation system, these keys could be uploaded to the device at installation time. To protect these initial keys against security attacks, it is important to avoid unauthorised modification (for further details see 4.2.1).

As mentioned in Section 2.3, such a centralised solution has a lot of disadvantages. The whole authentication process depends on the reliability of the key server. If the key server fails and an entity needs a new key, an authentication cannot be performed anymore (single point of failure). This means that the KDC must be fault tolerant and it must be carefully protected against security attacks. For example, if an attacker finds a way to gain unauthorised access to the key server, all stored keys are not secure anymore. Another problem is that every network node which wants to retrieve a key must establish a connection to the key server. If the amount of nodes and the session requests per time unit are very high , these connections could lead to a lot of network traffic. Therefore, these key servers are often bandwidth bottlenecks.

One possible solution is to replicate the key servers. Replication is often used to increase the fault tolerance and to distribute the incoming network traffic. Replication means the services or the data is redundantly available. In this case, replicating the key server means multiple copies of the server are available. This redundancy improves the fault tolerance. If one replica fails, another replica can assume the job. It is important to note that the protection against security attacks has not been improved. If an attacker gains unauthorised access to one of the replicas, the attacker has still access to the stored keys. As the attacker has more physical possibilities to attack the key server, replication makes the key server more vulnerable to security attacks.

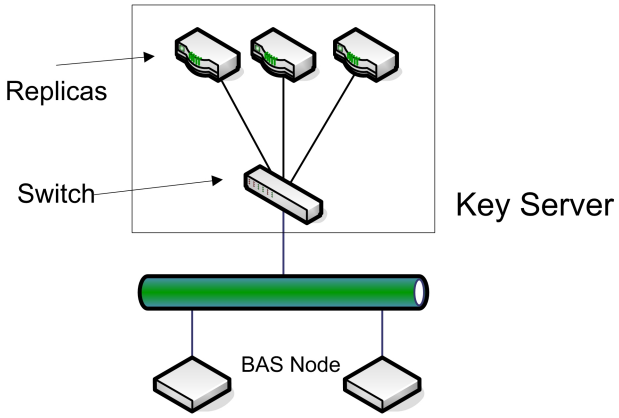


Figure 3.15: Replication of Key Servers

Another possibility is to divide the network into different domains. Each domain has its own key server and therefore each key server is responsible for the corresponding keys of its domain. These key servers are arranged in a tree-structured hierarchy. This means that if an entity wants to get the key of another entity that is not in the same domain, the key server forwards the request to the responsible key server of the foreign domain. To achieve this, each key server must know the public keys of its parents. Figure 3.16 shows an example, how such a hierarchy of key servers could look like.

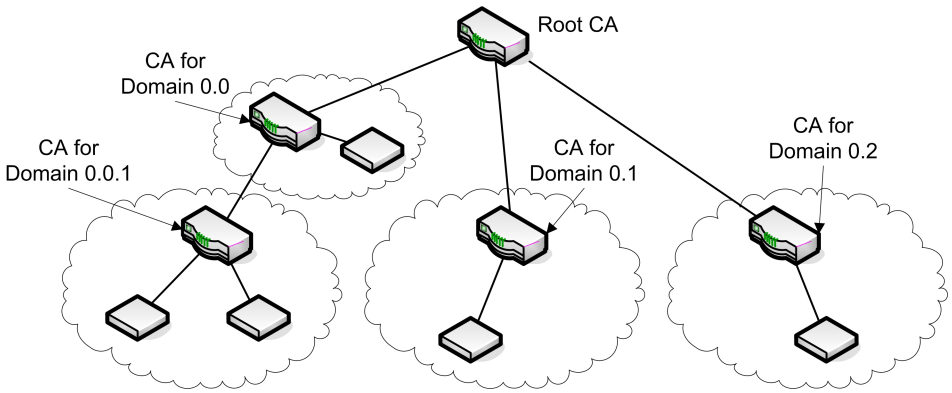


Figure 3.16: Hierarchical Organisation of Key Servers

The main benefit of this approach is that if an attacker gains unauthorised access to one of the key servers, only the keys of the particular domain are affected. This scheme of dividing the network into domains (Defence-in-Depth) is often used. For further details see Section 3.6.

All the solutions mentioned above use a central server (or a set of servers) to manage the keys. To avoid the disadvantages of these centralised solutions, a distributed approach can be used. An example of a distributed solution is a model called **web of trust**. In a web of trust no central key server is necessary to verify the authenticity of the required public keys. To achieve this approach, each entity has its own list of public keys which the entity trusts. This scheme works as follows. If an entity wants to set up a secure channel, it needs the public key of other communication participant. To get the public key, the entity first tries to find the key in its list of trusted public keys. If the list does not contain the required public key, the entity has to ask other network nodes. To avoid that an attacker responds with an invalid public key, the entity only asks network nodes that the sender already trusts. Suppose that an entity A needs the public key of B. A does not already have the public key, so it asks the trusted entity C whether C knows the public key of B. If C has it and A trusts C (this means that A already has the public key of C), A can set up a secure channel to C. Now, C is able to securely transmit the public key of B to A.

It is obvious that this approach is hard to realise. To get a particular public key, it is necessary to find a trusted entity which knows the key. This means that the time which is needed to find the public key is not bounded. Therefore, this web of trust approach seems less appropriate in building automation systems.

3.4 Security in Development Process

As mentioned at the beginning of this chapter, it is essential to implement security mechanisms to protect the system against security attacks. To achieve an effective protection, security must be incorporated into the development process. It is important to consider security in each phase of the development process, even at the beginning. Unfortunately, a lot of developers start to implement the functionality. If the implementation of the core components has been finished, different security mechanisms are added. This way of adding security features is not the best one.

3.4.1 System development life cycle

It is essential to include security considerations in each phase of the life-cycle of the development process. This **system development life cycle** is a formal approach for the development of computing applications. In [37], this life-cycle is divided into six phases. Figure 3.17 illustrates this classic approach.

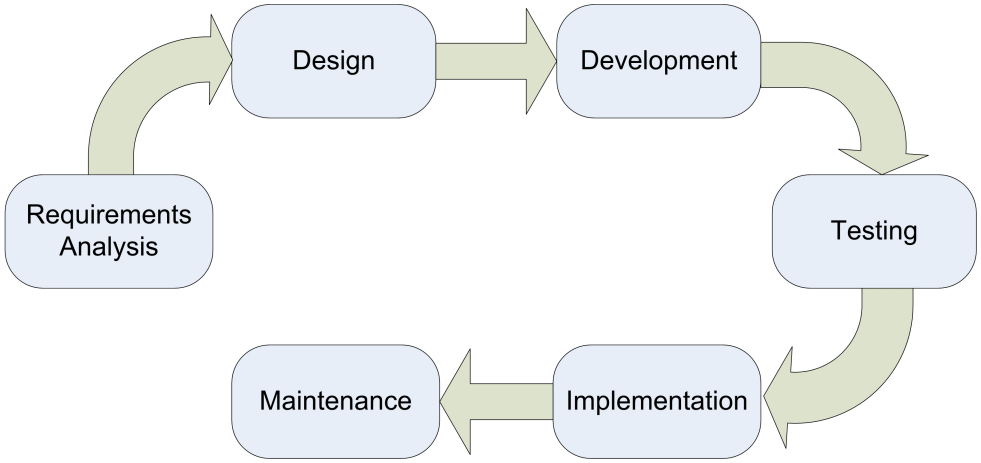


Figure 3.17: System development life cycle ([37])

In the first phase (**requirements analysis**), the application requirements must be figured out. This means that the objectives and the problems which the system has to solve must be analysed. In this phase, it is important to analyse the security requirements too. Therefore, the security objectives must be clearly identified.

As mentioned at the beginning of this chapter, this is done by defining a security policy. This security policy specifies which operations are allowed and which must be permitted. This policy is the base for further security considerations. In later phases, the security policy helps to verify whether the necessary security requirements have been satisfied.

In the **design phase**, the different possibilities to fulfil the application requirements are analysed. These different ways and mechanisms how the system can be realised must be compared to each other. After this comparison, the most efficient way which meets the requirements must be chosen. In this phase of the development process, the different security mechanisms must also be selected. Based on the security policy of the requirements analysis, available security mechanisms which meet the specified security requirements must be analysed. To achieve this, it is often advantageous to perform a risk analysis (for further details see [38]). This analysis of risks helps to find the best way that fulfils the specified security requirements.

After the mechanisms have been chosen, the application software is developed (**development phase**). It is important to distinguish between the development of software for building automation systems and the development of IT software (for example web applications, database software, office applications,...).

As mentioned in Section 2.5, a lot of field and interconnection class devices have little computing power and limited memory. Therefore, different development techniques are necessary. Nowadays, systems of the IT domain (for example workstations) have a lot of memory and a high clock rate. Therefore, the developers do not need to think about memory usage and runtime. At this higher level of abstraction, hardware considerations are less important. Most of the applications which are designed for building automation systems will run on embedded devices where it is important to take the algorithm complexity and the memory usage into account. The developers must have the necessary qualification and experience to design such software components for embedded devices.

To satisfy these needs, they must be familiar with the used hardware as well. The developers must know the relationship between hardware and software. The development tools (for example C compiler, debugger, ...) which were supplied by the manufacturer of the hardware are more or less mature. Therefore, it could be necessary to write parts of the software code in assembler. It is important to note that these low level development tools must be used with care. High level development tools have implemented special mechanisms that provide a higher level of abstraction. These mechanisms avoid typical implementation mistakes. For example, many higher programming languages use automatic memory management,

which does not support the usage of pointers. Therefore, implementation mistakes like buffer overflows are less likely. It is important to note that low level development tools which are used to develop embedded software do not avoid such implementation mistakes. Therefore, the software engineers themselves must avoid such vulnerabilities.

	Embedded Device	**Traditional System**
Hardware	low clock rate, low memory	high clock rate, high memory
Programming Language	Assembler, C, maybe C++	higher OO languages (C#, Java, ...)
Abstraction Level	low	high

Table 3.1: Differences in Development Process

In the **testing phase**, the developed software components are verified. It must be tested whether the application satisfies the specified requirements. Additionally, it must be verified whether the system fulfils the necessary security requirements which have been defined by the security policy. Generally, the members of the development team should not be responsible for performing these tests. A better approach is to use a separate testing team. If a tester is not directly involved in the development of the application, he will be unprejudiced. Therefore, it is more likely that one of these independent testers will find flaws and vulnerabilities. Additionally, a separate testing team can perform the tests without having the fear of having to change the system.

After the application meets all specified requirements, the system must be implemented (**implementation phase**). In the building automation domain, this phase is also referred to as system integration. In this phase, the project engineer must integrate the developed software applications as well as the hardware components. This means that the different network nodes must be installed and the software applications must be uploaded to the corresponding network devices. Additionally, the engineer must perform different configuration task. For example, an individual network address must be assigned to each network device. If the used security mechanisms need an initial setup, this setup must also be performed in this phase. For example, if an authentication mechanism that is based on a KDC is used, each node must get a secret key which it shares with the KDC. Therefore, at installation time, the project engineer must distribute these secret keys in a secure manner.

After the integration and the configuration of the entire system has been finish, a system test is necessary. This system test has the main objective to verify whether the system as a whole meets the specified requirements. Especially, the integrated security mechanisms must be tested whether they satisfy the security requirements that have been defined in the security policy.

After this integration, the system must be maintained in order to keep the it operable (**maintenance phase**). It is obvious that an application is never free of vulnerabilities. Therefore, it must be possible to correct these software bugs. Additionally, it is sometimes necessary to implement extra features which are not defined in the specification. To achieve these software changes, it is necessary to perform a software update (for further details see Section 3.5).

3.4.2 Design Issues

As mentioned in Section 2.1, a modern building automation systems (better a building automation network) is normally designed distributed. Compared to centralised solutions, the design of a distributed approach is more complex. Especially, the protection against security attacks can be more complicated if a distributed solution is used. Therefore, there are some design issues which help to make the system more secure.

To avoid unauthorised access, the particular service or data must be protected. [20] describes three different design principles how an entity can be protected against unauthorised access. First, the data or the service itself can be protected. This means that the entity itself avoids invalid operations. For example, a microcontroller of an embedded device can deny the write access of different memory blocks (read-only memory).

Another solution is to protect the entity against unauthorised requests. This means that the message interface of the entity verifies whether a request is valid or not. For example, in a fire alarm situation it must be avoided that someone activates the elevator. To achieve this, the message interface of the elevator device could inhibit such requests.

Another approach is to protect the entity against unauthorised users. With this scheme, the identity of the initiator must be verified. If the initiator proves its identity, the receiver of the request can determine whether the entity has the necessary access rights to perform the requested operation. For example, in a building automation system only the system operator is allowed to change configuration parameters of field class devices.

Figure 3.18 illustrates these three possible solutions.

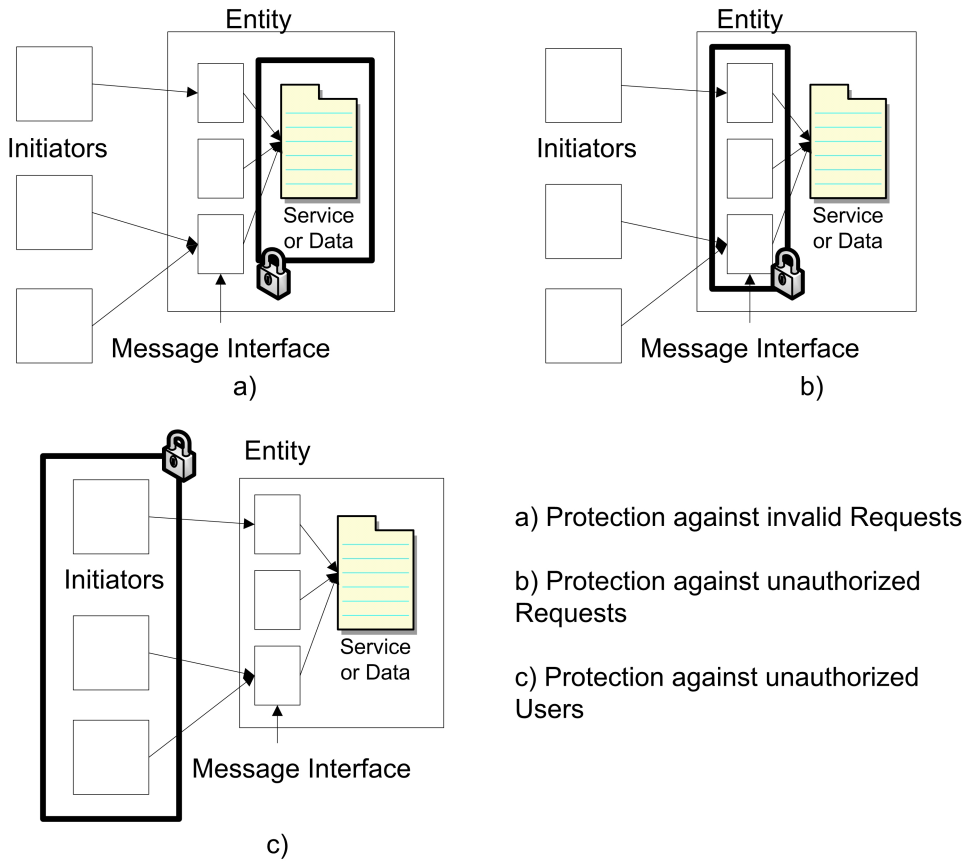


Figure 3.18: Avoidance of unauthorised access

There are several ways to make a system secure. First, it is possible to add a **security layer**. This layer is responsible for protecting of system against malicious attacks. It is important to note that the security of a system can only be guaranteed, if the mechanisms which the security layer relies on are secure. For example, consider a system where the security layer depends on the underlying operating system. The security of this system can only be ensured, if the operating system itself is secure.

Another important issue is the **weakest link**. A system is only as secure as its weakest link. Consider a system which uses strong security mechanisms. If it is possible to bypass these strong security mechanisms and if it is possible to find a way to gain unauthorised access without too much effort, the entire system is not very secure. If such a weak link exists, it is only a matter of time until an attacker

finds this weak link. For example, if the transmitted data is encrypted using a strong cryptographic algorithm, the messages are protected against interception but if the key is exchanged in an insecure manner (for example by transmitting the key in cleartext), the protection against interception cannot be guaranteed anymore. As a result of this, it is important to avoid all possibilities of violation of the security of the system.

In many building automation systems, special hard- and software is used. As the details about the hard- and software design is kept secret, the developers assume that nobody has detailed knowledge about the used technologies ("**security by obscurity**"). They think, without this knowledge, the attackers would not be able to find vulnerabilities. Such a conclusion is very dangerous. If vulnerabilities are not known, it does not mean that there are no vulnerabilities. For example, if a cryptographic algorithm is weak and insecure, it will not become securer, even if the source code is not available. It is only a matter of time until an attacker finds a vulnerability. For example, an attacker can use reverse engineering techniques to get the necessary information about the system. A better approach is to use security mechanisms which are public. This means that the security mechanism must not get weaker even if the source code is publicly available.

Another important issue is the use of **standard cryptographic algorithms**. Whenever it is possible, standard cryptographic algorithms should be used instead of newly developed ones. Developing secure cryptographic algorithm needs a lot of time, effort and expertise. Therefore, the development of new cryptographic algorithms should be avoided. As mentioned in Section 3.2.1, a lot of standardised cryptographic algorithm exists. These methods were developed and tested by experts who have several years of experience. Most of these standardised algorithms are public software. Therefore, it is recommended to use existing algorithms.

3.5 Update Policies

Building automation systems should run stable for many years or even decades. To achieve this, the system must be maintained in order to keep it operable. Since software is never error-free, it is possible that implementation mistakes and new vulnerabilities will be found. To correct these software bugs, it must be possible to install software patches. Sometimes, the operation environment changes and therefore it is necessary to add additional functionalities. So, it must be possible to add new software features without too much effort.

To satisfy these needs, an update mechanisms must be integrated. As a result

of the fast grow of the Internet, it is common nowadays to perform online updates. For example, an office workstation has normally a permanent connection to the Internet. Therefore, it is very easy to update such a workstation online. In a building automation system, this online update scheme is not always easy to achieve. It is obvious that it is not possible to connect each network node directly to the Internet.

One possible solution is to integrate a central server, which distributes the update packages to the different devices. This **update server** has the objective to distribute new software packages. For example, an IP gateway could act as such an update server.

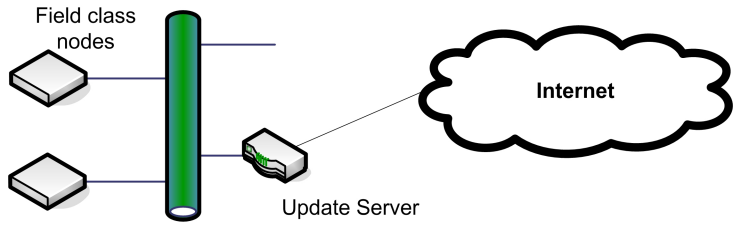


Figure 3.19: Online Update Server

As mentioned in Section 3.3.6, a centralised solution has a lot of disadvantages. For example, if a single server is used to update each device, such an update can be very time-consuming.

Therefore, it is more practical to use a distributed scheme. Similar to the key server approach (for further details see Section 3.3.6), each network domain has its own update server. Again, these update servers are arranged in a tree-structured hierarchy. On the top of this tree structure, the root update server is located. This server is connected to the Internet or another WAN to receive the new software packages. Figure 3.20 illustrates this concept.

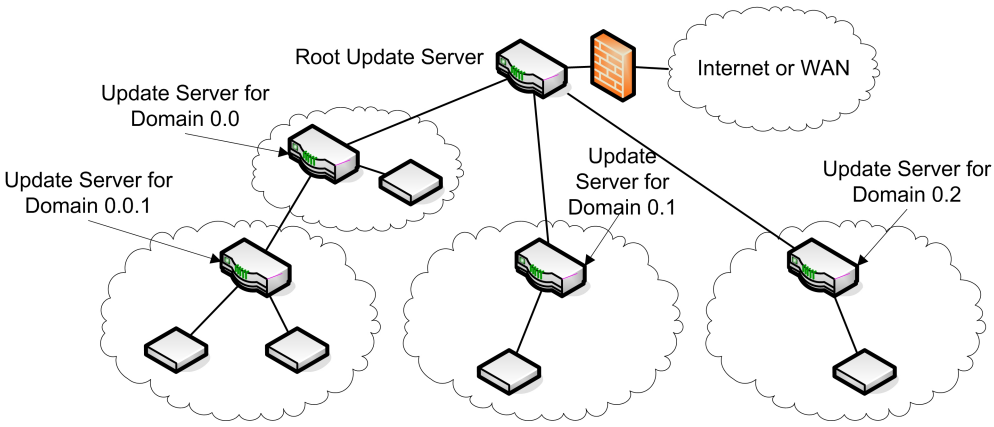


Figure 3.20: Hierarchical Organisation of Update Servers

3.6 Security Zones

In earlier days, building automation systems were isolated from other systems. An interconnection to other networks (for example to an office LAN or to another WAN) was uncommon. Some systems provide the opportunity to access the network via dial-up modems. These dial-up connection were used to maintain and configure the system remotely. It is obvious that these dial-up connections must protected against unauthorised usage. Otherwise, an attacker could take advantage of this dial-up mechanism.

Due to the assumed isolation, no protection mechanisms against security attacks were implemented. As long as an attacker has no physical access to the building, the attacker has no possibility to gain unauthorised access. As soon as the attacker has found a way into the building, there are no protection mechanisms that avoid a security attack (attack from the inside). Especially, on public places it is very difficult to protect the system against such attacks from the inside (for example, in a theater, shopping mall,...).

Nowadays, this isolation is no longer (if it ever was) valid. Modern building automation systems have an interconnection to other foreign networks. For example, an IP gateway provides the possibility to interconnect the building automation system with an IP network. This interconnection to other heterogeneous networks provides a lot of benefits. For example, an IP gateway could provide an HTML interface which can be used to maintain and to configure the system. It is important

to note that these gateways offer the possibility to attack the building automation system from a foreign network. This means that the attacker does not need to have physical access to the network. To avoid such attacks from ”the outside”, it is necessary to monitor and to restrict the access from such insecure networks.

There are two basic approaches to protect a network against security attacks from foreign networks. First, it is possible to put a single, insurmountable wall around the system. This approach is also called **hard perimeter**. Figure 3.21 illustrates this scheme.

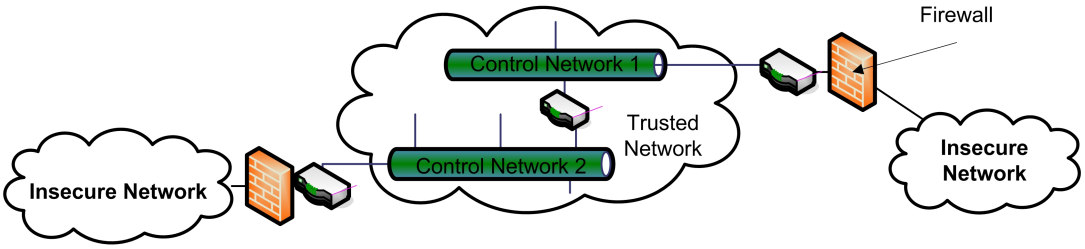


Figure 3.21: Hard perimeter approach

This security approach is often used to avoid unauthorised access from other networks. To achieve such a protection, a **firewall** is often used. Such a firewall has the objective to restrict the access to the internal network. It monitors and inspects the network traffic and verifies whether a connection request from a foreign network is allowed or not. Based on a set of rules, the firewall decides whether a connection is passed or rejected.

Depending on the way how the firewall inspects the incoming data packets, two different types of firewalls exist. The first type is called **packet-filtering gateway**. A packet-filtering gateway inspects only the header of the incoming and outgoing messages. Based on the header and on a set of specified rules, a packet-filtering gateway decides whether a packet is passed or dropped. For example, such a firewall could prohibit the access from certain devices. To achieve this, the firewall checks the source address of the messages which is encoded in the header of the network message. If the source address does not belong to the set of valid addresses, the particular data packets are dropped.

The second type of firewall is called **application-level gateway**. These gateways inspect the content of the data packets as well. Application-level gateways are more flexible than packet-filtering firewalls. Since these firewalls inspect the

content too, more complex rules can be defined. For example, in a building automation system, the access to the system via a foreign network can be limited. The application-level gateway could allow performing monitoring task whereas it could discard requests which have the objective to change configuration parameters.

As mentioned before, application-level gateways are more powerful. Compared to packet-filtering gateways, more complex rules can be defined. To get the necessary information, it is necessary to have detailed knowledge about the used protocols. Sometimes it is even necessary to observe the current internal state of a connection (for example, whether a connection has already been established or authentication has been performed or not). As this internal state must be logged, these gateways need more memory. In addition to that, the inspection of the message content requires more time. Since interconnection devices of building automation systems have limited hardware resources, it could be possible that these devices cannot fulfil these demands.

The hard-perimeter approach mentioned above has a lot of disadvantages. This scheme only protects the network against malicious attacks from the outside. It does not avoid security attacks from the inside. Behind this single wall of protection, no more security mechanisms exist. If the attacker bypasses this security wall, the attacker has gained unauthorised access to the system. Additionally, it is very unrealistic that this single wall protects the system against all kinds of attacks. Even the best designed system has its weaknesses. So, it is only a manner of time before an attacker finds a vulnerability. Therefore, it is very dangerous to rely on this single protection mechanism. If this single wall fails, the attacker has access to the entire system.

A more secure solution is an approach called **defence-in-depth**. In the defence-in-depth approach, the system which should be protected against malicious attacks is divided into several security zones. Each zone has different types of security mechanisms implemented. It is important to note that these different zones can be ordered hierarchically. This means that it is possible to define outer and inner zones. The outer zones contain less valuable devices, whereas critical parts of the systems are located in the innermost zones. This concept is illustrated in Figure 3.22.

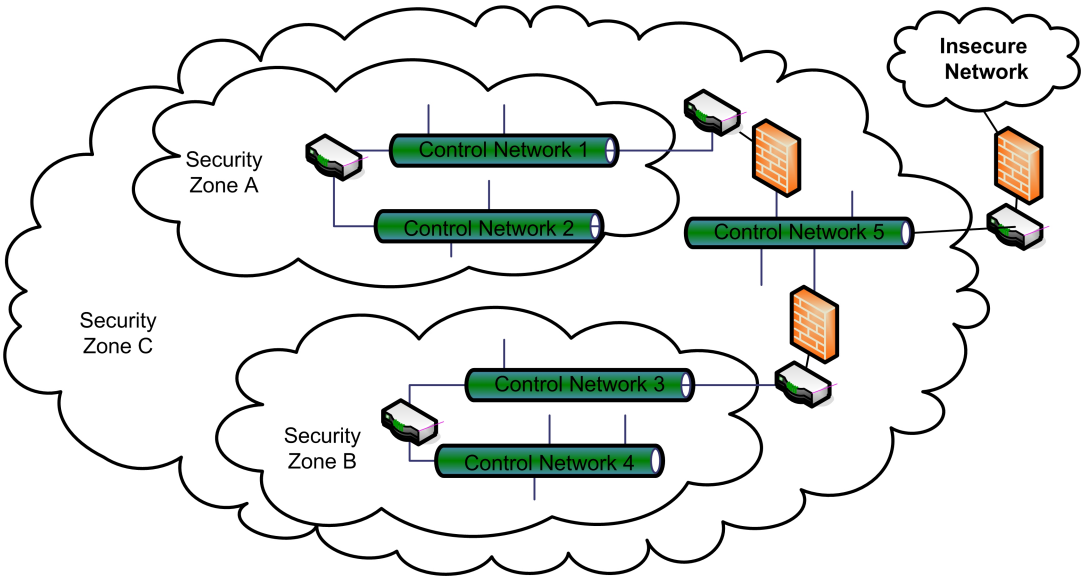


Figure 3.22: Defence-in-depth

In [40], a few defence-in-depth principles are described. The first one is called layered defence. In the defence-in-depth concept, the different security zones are like shells which are placed around the system. This means that if an attacker bypasses the first security layer, a second one still protects the critical parts of the system. It is important to note that an attacker from the outside can only see the first security layer. As long as this first layer does not fail, the second defence mechanism does not come into play. So, the attacker cannot begin to attack the security mechanisms of the inner zones.

This approach has another important benefit. If an attacker has already access to the outer security zone (for example, the attacker uses a valid user account), only the resources inside this zone are affected. More critical parts of the system are placed in another zone which has additional security mechanisms implemented. As long as there is still a security layer between the attacker and the resource, the resource is protected against attacks even from the inside of the system.

To achieve this layered defence, different security zones implement different security mechanisms. Therefore, it is common to define a separate security policy for each security zone. It is essential to clearly define the demands on each zone. It must be determined for example which parts of the system must be placed into

inner zones and which devices can be located in the less valuable, outer zones.

It is also important to implement protection mechanisms on multiple places. This means that all different possibilities to gain unauthorised access must be avoided. Consider, for example, a building automation system which has an interconnection to an IP network. Suppose, the IP gateway is carefully protected against malicious attacks. As long as there are no other ways to gain access to the system, the system seems to be secure. Unfortunately, if there is an additional connection to the outside (for example in the case of an IP outages a dial-in modem can be used to maintain the system remotely) which is not sufficiently protected, the security of the system cannot be guaranteed anymore. It is essential that this way is also carefully protected (even if it is used infrequently). Otherwise, it is only a matter of time until an attacker uses this way to gain unauthorised access.

To provide an effective protection against security attacks, defence-in-depth solutions should also support mechanisms to detect an intrusion. These intrusion detection mechanisms (for further details see 3.7) are necessary to inform other security layers that an attack is occurring. If such intrusion detection mechanisms are not integrated, the different zones are independent from each other and so the intruder can attack each security layer independently. With intrusion detection systems, a security attack can be recognized and defensive reactions can be performed. More details about such intrusion detection systems are explained in the next section.

The usage of defence-in-depth is not restricted to networks. It is also possible to use defence-in-depth for software applications. This means that the software itself has implemented several security layers. Consider, for example, a software implementation which provides a message interface to change the content of a database. This message interface itself verifies the incoming requests and checks whether they are valid or not. The underlying operation system has implemented additional security mechanisms (for example a memory protection) and therefore it provides a second security layer (for further details on defence-in-depth of software applications see [41]).

3.7 Intrusion Detection

It is obvious that each system has its weakness. So, it is always be possible that there is still a way left open to gain unauthorised access. To rely on the assumption that nobody will ever find a vulnerability is very dangerous. Therefore, a protection against security attacks is not always enough. It is also important to im-

plement mechanisms that discover intruders. **Intrusion detection** does not only include the detection of attackers that have already gained unauthorised access. It is also essential to discover abnormal activities which indicate a security violation. As mentioned in the previous section, if a defence-in-depth approach is used, such an intrusion detection system (IDS) must be integrated.

There are two different kinds of intrusion detection systems. Depending on the observed activities, host-based and network-based intrusion detection systems exist (see Figure 3.23).

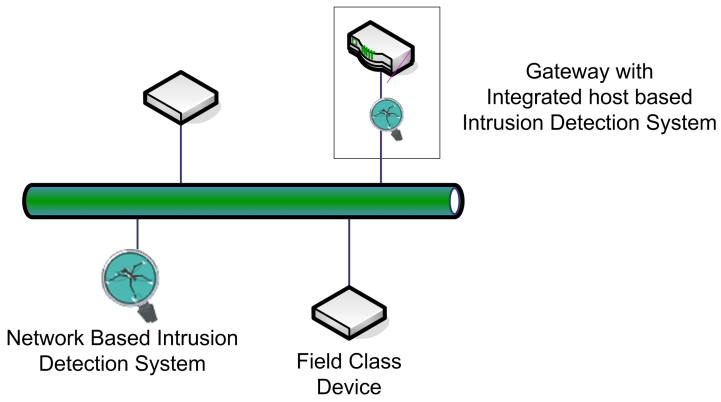


Figure 3.23: Network-based and host-based intrusion detection systems

A **host-based intrusion detection system** tries to discover abnormal activities on a single host. Host-based intrusion detection systems can use different possibilities to detect an intruder. In [38], four different methods are described. Three of these methods are suitable in building automation systems as well and therefore these three will be discussed in the following paragraphs.

First, it is possible to observe the actions which are performed by an entity. These **user profiling** methods are based on the fact that each user has a characteristic profile. For example, a user normally accesses the same files, uses some commands more than others and so forth. An intruder can be detected by comparing the current activities against the expected ones. Consider, for example, a field class device in a building automation system which acts as a light switch. If someone uses this device to perform abnormal operations (for example an attacker uses the device to open a door), these ominous actions indicate that something is going wrong.

Another kind of host-based intrusion detection is called **intruder profiling**. Similar to user profiling, intrusion detection systems which use intruder profiling methods try to detect activities which an intruder normally performs. This means that the intrusion detection system searches for actions which are characteristic for an intruder (for example abnormal memory access).

As mentioned before, host-based intrusion detection systems work on a single host. Especially, security-relevant and important hosts of the network should be secured by such a host-based intrusion detection system. In a building automation system, interconnection class devices can be such important network nodes. For example, it could be advantageous to integrate an intrusion detection system in a gateway that interconnects the building automation system with a foreign network (see Figure 3.23).

Host-based solutions have one disadvantage. They can only observe the activities which are performed on a host. A **network-based intrusion detection system** observes the network traffic and therefore these systems are able to discover abnormal activities which do not only affect a single host. Consider, for example, an attacker that wants to gain access to a field class device of a building automation system. To avoid that the attack raises suspicions, the attacker tries only one password each time (for example a default password or the empty password). If the attempt was unsuccessful, the attacker tries to connect to the next device. A host-based intrusion detection system is not able to detect this attack because a single request does not indicate a security attack. On the other hand, a network-based intrusion detection system is able to detect these ominous requests by monitoring the network traffic.

In a building automation system, such a network-based intrusion detection system can be used to observe the network traffic. For example, such a system can discover abnormal management communication. As mentioned in the previous sections, an avoidance of Denial-of-Service attacks can be very difficult. A possibility to detect Denial-of-Service attacks is the use of an intrusion detection system. If a Denial-of-Service attack is detected, the intrusion detection system can try to cut off the network segment that is the source of the attack (for further details see Section 4.2.3).

Chapter 4

Security Threats and possible Solutions in BAS

In the previous chapters, basic terms and definitions were explained. In Chapter 2 a brief introduction into building automation systems was given. Chapter 2 explained what a building automation system is and gave an overview about the most important concepts. In the following Chapter 3, basic security mechanisms and concepts were discussed. Chapter 3 gives an overview about the existing security mechanisms. Additionally, their importance for building automation systems was discussed.

Now, it is time to take a closer look at the different security threats in a building automation system. As mentioned in Section 2.3, in a building automation system, the functionality is normally implemented using a two-level architecture. This two-level model consists of a control network and a backbone network. Based on this two-level model, an intruder has two possibilities to attack a building automation system.

On the one hand, it is possible to attack the building automation system from a foreign network. This means that the attacker tries to gain unauthorised access to a device which interconnects the building automation system to a foreign WAN (for example with the Internet). Additionally, it is also possible to attack the backbone network itself. These attacks are also called **attacks from the outside**. These attacks are discussed in Section 4.1.

On the other hand, it is also possible to perform an **attack from the inside**. To achieve such an attack, the intruder can attack the control network itself or he can attack a field class device. In both cases the attacker must have physical access to the control network. Attacks from the inside are discussed in Section 4.2.

Figure 4.1 summarises these types of threats. It is important to note that a device which interconnects the control network with the backbone network can be attacked from both sides. First, it can be attacked from the control network. This type of attack will be discussed in Section 4.2.1. A device can also be attacked from the backbone network (see Section 4.1.1). The rest of this chapter will take a closer look at these different possibilities how a building automation system can be attacked.

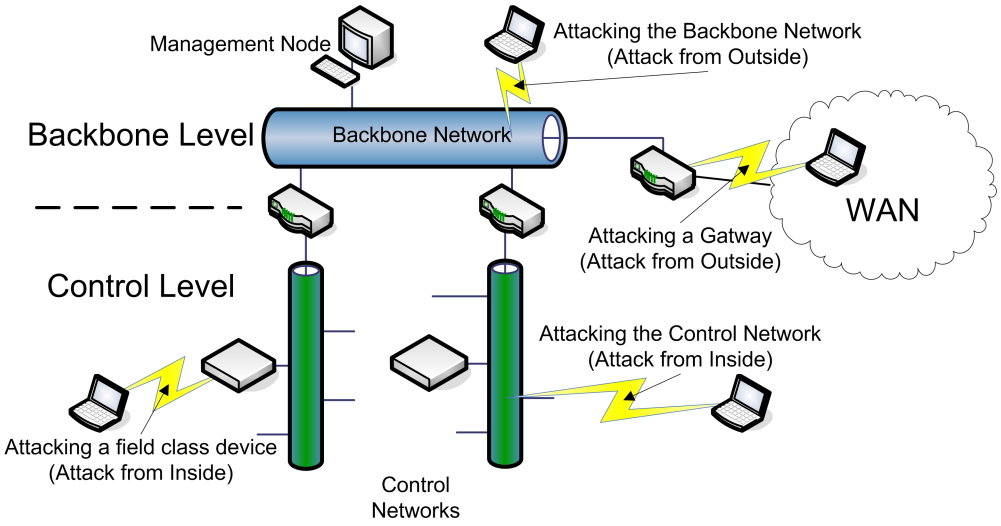


Figure 4.1: Possible Threats in a Building Automation System

4.1 Threats from the Outside

As mentioned above, attack from the outside means the attack is performed from a foreign network. Due to the fact that the attacker does not have physical access to the network, the attacker must find a way to gain access to the internal building automation network.

To achieve this, the attacker has two possibilities. First, the attacker can try to attack the gateway which interconnects the building automation network to the foreign network. This possibility is described in the next Section 4.1.1.

Another alternative approach is to attack the backbone network itself. Especially, if a tunneling mechanism (for further details see Section 2.3) is used, an

attacker could try to attack the logical tunnel. This kind of attack is explained in Section 4.1.2.

4.1.1 Attacking Interconnection Devices

Modern building automation systems normally have an interconnection to a foreign network (for example to a WAN like the Internet). With this interconnection, it is possible to maintain and configure the system remotely. For example, a webserver could provide an HTML interface which can be used to monitor the building automation system. To provide such a connection, an interconnection device is necessary. This device handles the interconnection between the foreign network and the building automation network. If the network protocols are not identical, such an interconnection device must perform a protocol conversion between the foreign network and the building automation network.

It is obvious that these interconnection devices must be carefully protected against security attacks. If the building automation system is isolated and the system can only be accessed through such a device, the attacker has to find a weakness in the implementation of the interconnection device. Therefore, several security mechanisms are necessary to protect an interconnection device against malicious attacks.

Based on [12] and [10], at least three security mechanisms are necessary to protect an interconnection device against security attacks. These three mechanisms are:

- Access Protection
- Authentication and Authorisation
- Securing the connection to the gateway

First, it is necessary to protect an interconnection device against unauthorised access (**access protection**). The most commonly used method is to use a packet-filtering firewall. As mentioned in Section 3.6, such a firewall provides only basic protection. Based on a set of rules, a packet-filtering firewall decides whether a data packet is passed or dropped. Therefore, such a firewall is not always sufficient.

In [12], a more secure approach is explained. This **remote login scheme** works as follows. Suppose, a remote host wants to perform certain tasks remotely. First, the remote host establishes a secure connection to the first dedicated machine. This machine is responsible for verifying the identity of the remote host. Therefore, the first dedicated machine sends a request for authentication. After the remote host has proven its identity, the first dedicated server requests the second

dedicated server to establish a secure connection to the remote host. It is important to note that this second server only accepts connections from the first dedicated server. All other incoming request are dropped and so a remote host cannot open a connection to this second server directly. After the first dedicated server sends a request to the second one, the second server establishes a secure connection to the remote host. Since this second dedicated server has a permanent connection to the gateway behind the firewall, the second dedicated server acts as a proxy for the gateway behind the firewall. The second dedicated server forwards the incoming traffic to the gateway and so, the remote host is able to communicate with the gateway. The gateway behind the firewall only accepts connections from the second dedicated server. Again, all other incoming requests are dropped.

It is important to note that neither the gateway nor the second dedicated server can be accessed directly. The main advantage of this scheme is that a remote host can only access the first server directly. As the second dedicated server only accepts connections from the first dedicated server, the second server is not accessible from the foreign network. This means that the second server is invisible to the foreign network. Figure 4.2 illustrates this approach.

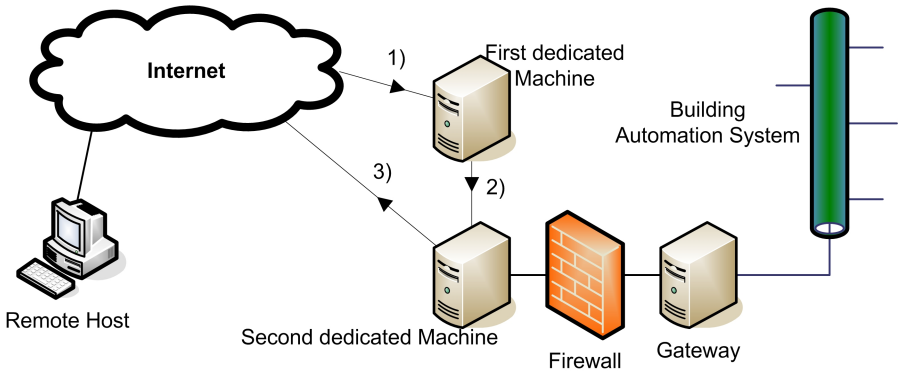


Figure 4.2: Remote Login Scheme

Anyhow, if it is still possible to gain unauthorised access, the whole building automation system is affected. This means that if an attacker bypasses this single wall of protection, the intruder has full access to the whole system. To avoid this problem, a **defence-in-depth** approach can be used. As mentioned in Section 3.6, in the defence-in-depth approach, the network is divided into several security zones. If such a scheme is used, the interconnection device that provides a connection to a foreign network should be located in the outermost zone. The main

benefit of this concept is that critical parts of the system which are placed in inner zones are still protected, even if an attacker has successfully gained access to the interconnection device. In Section 6.2.2 a secure architecture called EIBsec is explained. This architecture shows how a defence-in-depth approach can be used in a building automation system.

One of the most important mechanisms to protect the interconnection device is **authentication**. The interconnection device must determine whether the remote host is allowed to initiate the requested operations. To achieve this, it is necessary to verify the identity of the initiator (client authentication). Otherwise, an attacker could perform illegal actions. It is also necessary to verify the identity of the interconnection device (server authentication). The remote host must be sure that the received responds are really coming from the interconnection device. Otherwise, an attacker could intercept the request from the remote host and send a faked respond back to the host. Consider, for example, a security alarm system which provides an HTML interface. With this HTML interface, a remote user is able to monitor the building. To avoid an unauthorised use of this HTML interface, the access to this interface must be restricted. Therefore, the remote user must prove its identity (client authentication). Additionally, the webserver must also prove its identity (server authentication). If the identity of the webserver is not verified by the remote user, an intruder is able to pretend that everything is ok. So, the attacker is able to hide an intrusion.

To verify the identity of both communication participants, an authentication protocol must be implemented. As mentioned in Section 3.3, many different authentication protocols exist. Due to the fact that the attacker does not need to have physical access to the building automation system, the attacker can use its own hardware resources to perform a malicious attack. This means that the attacker has enough time and performance to find valid secret keys or passwords. Since the remote host and the interconnection device have normally enough computing power and memory, a strong authentication mechanisms should be chosen (for example an authentication protocol that uses a strong cryptographic algorithm and long keys).

After the identities of both communication participants have been verified, it must be determined whether the remote host has the necessary access rights. To achieve this, an **authorisation** mechanism must be implemented. In Section 3.3.5, different possibilities were explained how such a mechanism can be realised.

Independent of the chosen authentication and authorisation protocols, three different possibilities to provide authentication exist. First, only the interconnection device verifies whether the remote user is allowed to perform the requested

operation. This means that the interconnection device is responsible for verifying the identity of the remote host. Once the identity has been proven and the remote user has the necessary access rights, the request is forwarded to the particular device in the internal network. The main drawback of this method is that the building automation system is not protected against unauthorised access from the inside.

Another possibility is to use a decentralised approach. This means that the field device itself has to verify the identity of the remote host. Using this approach, the interconnection device forwards the requests directly to the particular device. It does not verify whether the remote user is allowed to perform the requested operation. The field device itself must verify whether the remote host has the necessary access rights. The main disadvantage is that each device must maintain its own access control list. Additionally, if a cryptographic algorithm is used, the device must store a certain number of keys. Since field class devices normally have limited memory and limited computing power, this approach is typically not suitable for use in building automation systems.

A better approach is to use a gateway as a proxy. This scheme works as follows. First, the remote device establishes a connection to the gateway. The gateway verifies the identity of the remote host and checks whether the host has the required access rights. After the request has been verified, the proxy performs a conversion of this request. Then, the proxy establishes a connection to the particular device and forwards the converted request to it. To be sure that the communication participants are what they claim to be, the proxy and the field device prove their identities. It is important to note that there is no direct connection between the remote host and the field device. The main benefit of this approach is that the proxy maintains all remote connections. The proxy decouples the two networks. Therefore, the field devices of the building automation system do not need to worry about remote users and hosts. Figure 4.3 illustrates this approach.

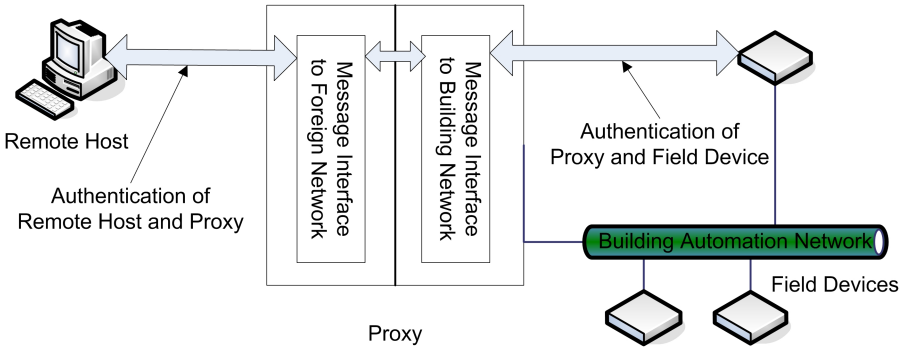


Figure 4.3: Proxy

It is obvious that an authorisation mechanism is not enough to protect the communication between the gateway and the remote host. Therefore, it is necessary to **secure the transmission channel** between the remote host and the gateway. To achieve this, it is necessary to implement a security protocol which provides a secure channel between the remote host and the gateway. As explained in Section 3.2, several mechanisms are possible to guarantee that a transmission channel is protected against security attacks.

A common approach is to use an existing security protocol. Due to the fact that these interconnected networks are often IP networks, IP based security protocols like SSL/TLS[1] and VPN[2] can be used. These protocols are placed on the top of IP or TCP. With these protocols, it is possible to communicate through a secure channel (for further details see [42] and [43]). For example, in [45] the use of TLS to protect the remote field bus access is discussed.

4.1.2 Attacking Backbone Tunnels

A common approach is to use a backbone network to interconnect different control networks. At the intersection points, interconnection devices are located. These devices are responsible for routing the network traffic between the backbone network and the different control networks. If the connected control networks use different network protocols, a protocol conversion is necessary.

As mentioned in Section 2.3, a scheme called tunneling is frequently used. With this approach, all protocol layers of the control network are transmitted over

[1] Secure Sockets Layer/Transport Layer Security
[2] Virtual Private Network

the backbone network. To achieve this, a logical tunnel is established between the interconnection devices of the control networks.

It is very popular to use an IP based network as backbone network. For example, an already existing office LAN can be used as backbone. This means that the backbone network must share the network resources with other systems (for example with an office LAN). Therefore, it is obvious that this logical tunnel must be protected against security attacks. It must be avoided that somebody intercepts or modifies the data packets which are transmitted through the tunnel. Additionally, an attacker must not be able to insert additional data packets (for example replaying previously sent messages). Therefore, a **secure tunnel** is necessary.

Figure 4.4 shows a possibility how such a tunnel can be attacked. Since the attacker is located between the two communication participants, this kind of attack is called **Man-in-the-Middle attack**. Consider, for example, a security alarm system that has the main objective to detect unauthorised intrusion. To achieve this, several glass break sensors are installed. If such a sensor indicates a broken window, it sends a "glass broken" message to the alarm annunciator (for example a signal lamp). As this device is not located in the same network segment, it is necessary to transmit this message over the backbone network. To hide the burglary, the intruder could perform a Man-in-the-Middle attack. This means that the intruder intercepts the "glass broken" message. To fool the alarm system, he sends a "glass OK" message to the alarm device. Without any security mechanisms, the alarm device is not able to detect an unauthorised modification of the received messages.

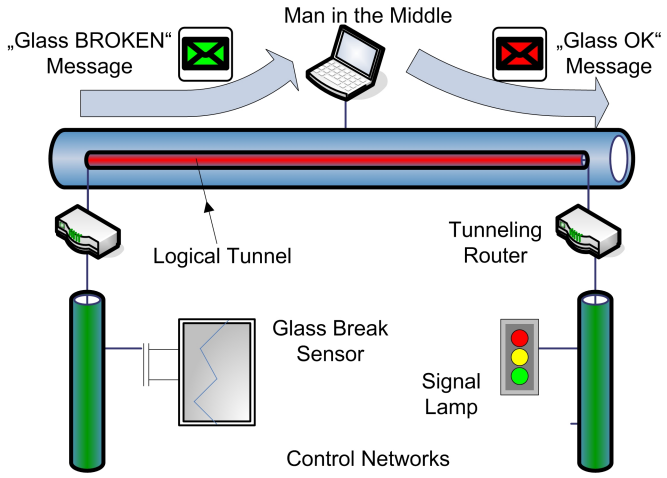


Figure 4.4: Man in the Middle Attack

To achieve a secure tunnel, several security mechanisms are needed. As mentioned in Section 3.2, it is necessary to protect the tunnel against unauthorised interception, modification and fabrication. In Section 3.2, different mechanisms were discussed which can be used to provide a secure tunnel. Again, if the backbone network is publicly accessible, strong cryptographic algorithms should be used. Therefore, the tunneling router must have enough computing power and memory to be suitable for using these algorithms.

Nowadays, many buildings have already an IP network (for example an office LAN) installed. Therefore, in a building automation system, it is advantageous to use this existing network as backbone network. So, it is obvious to use existing tunneling techniques which are based on IP (for example VPN). It is important to note that these IP based technologies were not developed for the use in the automation domain. This means that technologies like VPN do not satisfy the necessary requirements.

In [6] and [5], the problem mentioned above has been discussed. In the automation domain, a VPN solution does not always fulfil the necessary needs. VPN was designed for the use in networks where real time, safety and security requirements are more moderate. Therefore, the term **Virtual Automation Network (VAN)** has been defined. A VAN is more than a VPN. A VAN is a heterogeneous network which consists of different kinds of networks (for example the Internet or other wired or wireless LANs).

Compared to a VPN solution, the demands on a VAN are stronger. As men-

tioned in [6] and [5], in the automation domain, it is often necessary to fulfil real-time requirements. Ethernet-TCP/IP solutions (like VPN) are often not suitable for these real time requirements. Especially, hard real time constraints cannot be guaranteed. As mentioned in Section 2.4.1, in a building automation system, these real time requirements are more moderate. Anyway, they can still exist and therefore they must be taken into account.

Additionally, in a VAN, the safety and security requirements are stronger. Therefore, existing solutions are often not sufficient. To satisfy these needs, other solutions have to be found. For example, it is more secure to use a defence-in-depth approach instead of a traditional hard-perimeter one. For further details see [6] and [5].

4.2 Threats from the Inside

Another possibility to attack a building automation is to gain physical access to the control network. In the past, building automation systems were isolated from other, foreign networks. The system designers thought that as long as the physical access to the network is prohibited, the system is secure. So, no special security mechanisms were integrated.

This assumption ("security by obscurity") is very dangerous. Even if the access to the building is restricted, there are still possibilities to gain physical access to the building automation system. Especially, if a wireless medium or powerline is used, it is very easy to listen to the network without physical access. It is important to note that even if a wired medium like twisted pair is used, there are still possibilities to get access to the network medium.

One possibility is to use **social engineering** techniques. These techniques try to fool employees and legitimate users to get confidential information about the network. For example, an attacker could claim to be an administrator. If the identity is not verified and the access to the building is granted, the attacker has gained physical access.

At **public places**, a physical access to the network is easier to achieve. Since these places are public accessible, the access is not restricted and so, it is very easy for an attacker to gain access. It is obvious that it will attract attention, if someone tries to gain physical access to the network medium. For example, a clever hacker could develop a special wireless device. Such a "**hacker device**" (see Figure 4.5) can be attached to the bus. This special device acts like a "mini gateway". On the one side, it is attached to the control network of the building automation system

and on the other side the hacker can establish a remote, wireless connection to this device. With this wireless connection, the attacker is able to sniff and send messages remotely. The attacker does not need to be physical in front of the device or in the same room. It is very important to keep the ”hacker devices” smart. If it is small enough, the chance that someone recognises it is smaller. As this example shows, physical security is hard to achieve. This does not only apply to public places. Even in restricted areas, an attacker will find a ways to install such a ”hacker device”.

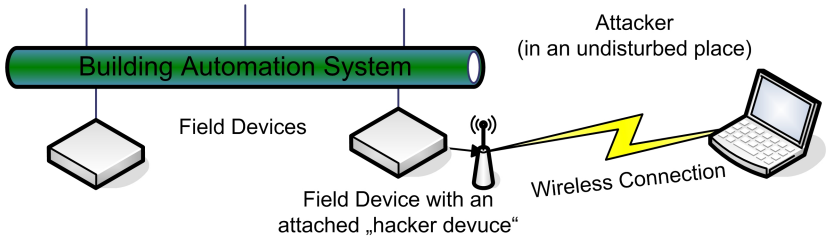


Figure 4.5: Attack with a wireless hacker device

As shown in the examples mentioned above, attacks from the inside are at least as dangerous as attacks from the outside. Therefore, the rest of this chapter takes a closer look at these kinds of attacks. Again, there are two possibilities. First, the field device itself can be attacked (Section 4.2.1). Second, an attacker could try to gain physical access to the control network medium. This possibility is explained in Section 4.2.2.

4.2.1 Attacking BAS Nodes

If an attacker has gained physical access to the building, the intruder is able to try to attack the control network.

One possibility is to try to attack a field class device. If the attacker has such a device under its control, the attacker has gained unauthorised access to the control network. This means that the intruder is able to perform further security attacks.

To avoid this, the field nodes must be protected twice. Most field class devices provide two external communication interfaces. Through these interfaces, it is possible to communicate with the device. Since normally a control network interconnects the different field class devices, field nodes have a **network interface**. This interface provides the opportunity to communicate with the device remotely.

For example, to configure or to maintain the device remotely, a management node can establish a management connection to the device.

Most field class devices have a second, **local interface**. Through this interface, it is possible to connect the device to another external device. For example, a sensor can be directly connected to a field device via a standard 4..20 mA interface. Another example of a local interface is a RS232 interface which can be used to connect the field device to a management node directly. It is important to note that even if the sensor is fully integrated into the containment of the field device, there is still an interface between the raw sensor and the microcontroller. Figure 4.6 shows these different kinds of interfaces.

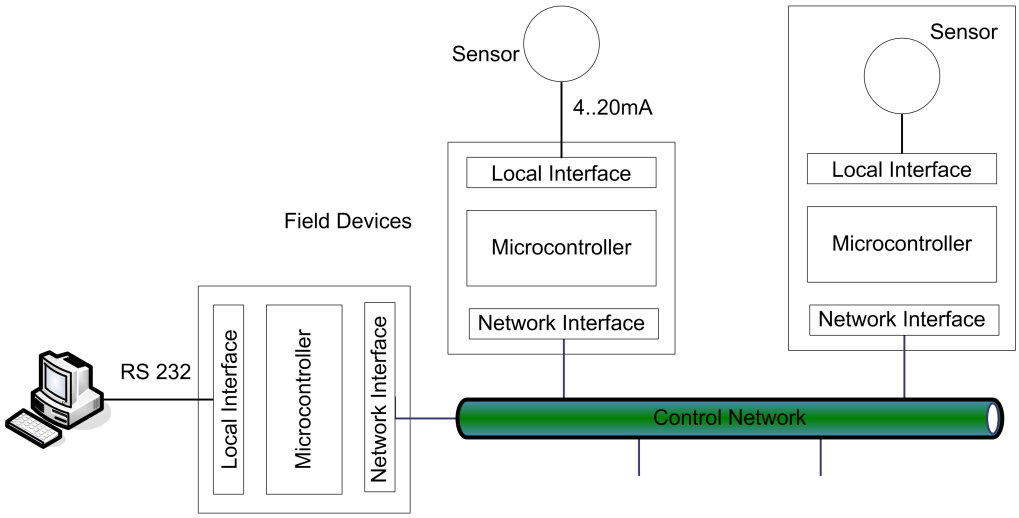


Figure 4.6: Different field device interfaces

To gain access to such a field device, the attacker has two possibilities. First, the attacker can try to attack the device remotely. This means that the attacker attempts to gain unauthorised access through the network interface. To achieve this, the attacker must have access to the control network medium. This access can be performed by attacking the control network itself (for further details see next Section 4.2.2) or by attacking a gateway (for further details see Section 4.1.1). To avoid such attacks, it is necessary to protect the network interface (**network protection**).

It is also necessary to avoid that an attacker is able to gain physical access to the device (**physical protection**). This means that an unauthorised person must

not be able to modify the containment of the device. Otherwise, the attacker is able to communicate with the device through the local interface. Consider, for example, a sensor which is connected to the field device through a local interface. It must be avoided that an attacker intercepts or modifies the raw sensor data that the sensors transmits to the local interface. Another example is a field device that provides a local management interface (for example RS232). Through this interface, it is possible to configure or maintain the device. It is essential to protect this interface against unauthorised usage. Otherwise, an attacker could use this local interface to perform management operations (for example changing configuration parameters).

To avoid such physical access by an unauthorised user, it is necessary to enclose the hardware in a **safe containment**. This containment must be stable enough to avoid easy, physical access. Additionally, seals or sensors can be used to detect attempts to damage the containment[3].

The field device must only be accessible through well-defined interfaces. These interfaces (both, local and network interfaces) must be protected against malicious attacks. To achieve this, an authentication and authorisation mechanism must be integrated. Such a mechanism is responsible for verifying the identity of the entity which wants to access the interface. Additionally, it must be checked whether the entity has the necessary access rights to perform the requested operation. Consider, for example, a management system which provides the opportunity to access the memory of the microcontroller directly. This management system provides commands like `Memory_Write()` or `Memory_Read()`. With these commands, it is possible to modify or read out the content of the of the memory. It is obvious that it must be guaranteed that only authorised entities are allowed to use these memory manipulation commands. Otherwise, an attacker could simply read out confidential memory content (for example secret keys or passwords).

In Section 3.3, several authentication mechanisms were discussed. These methods have one common drawback. All devices need an **initial secret** that must be known at installation time. For example, if a password based approach is used, each device must have an initial password that can be used to authenticate the first time. To solve this problem, default passwords can be used. It is important to note that these default passwords must be changed as soon as possible. Otherwise, an

[3]For example some EIB/KNX field devices provide the opportunity to detect a removal of the application module. If an attacker removes the application module the device sends a special group message which indicates the unauthorised removal.

attacker can use such well-known default passwords to gain access. If a KDC or a public key based authentication method is used, every device must have an initial set of keys. One possible solution is to integrate the required keys into the binaries of the application.

Furthermore, it is sometimes necessary to initialise the device. To achieve this, it must be possible to **reset** the device. It is important to note that an attacker can use this reset mechanism for his own purposes. Consider, for example, a mechanism that resets the password to the default value. If an attacker knows this default password, the attacker can simply perform a device reset. After this reset, the default value is set and so the attacker can gain access to the device. To avoid such an attack, it is essential to prohibit an unauthorised activation of this reset mechanism. For example, if the reset button is located at the front of the device, even the strongest security mechanism becomes useless. A possible solution is to put the reset button into the containment. If the containment is protected against unauthorised, physical access (for example by locking it), an attacker cannot use the reset mechanism to clear passwords or other secrets.

As mentioned in Section 2.5, field class devices have limited system resources. Compared to normal workstations, field devices contain low cost microcontrollers. Therefore, these microcontrollers have limited memory and limited computing power. Additionally, field devices often use a 8 or 16-Bit microcontroller. It is obvious that computationally intensive algorithms cannot run on such small embedded devices. In [46] and [47], the performance of the most important cryptographic algorithms was estimated. The authors of these papers performed different tests. They analysed different symmetric and asymmetric algorithms (RSA, triple DES, AES), the Diffie-Hellman key exchange algorithm and the hash functions MD5 and SHA1. These tests were performed on 8 and 16-Bit CPUs. Appendix A lists the results of these tests.

The results of these performance tests show that asymmetric algorithms are too slow to run on small embedded devices. Especially, the decryption process takes several seconds or even minutes. The usage of such an asymmetric algorithm leads to a delay which is not acceptable in the building automation domain. For example, if an asymmetric algorithm like RSA is used to encrypt the process data communication between the light switch and the light source, it will take several seconds or even minutes until the light is really turned on.

Therefore, if the hardware resources of the field devices are limited, a symmetric algorithm must be used. As shown in Table A.4 in appendix A, AES is much faster than triple-DES. Since AES is at least as secure as triple-DES, AES should be always preferred.

If the available memory and computing power are still insufficient, another solution has to be found. One possibility is to use a separate dedicated hardware component which acts as an en-/decryption unit (Figure 4.7). Such a unit is attached to an existing microcontroller and has the main aim to encrypt and decrypt all data packets. The sensor sends the raw sensor value to the microcontroller. The microcontroller takes the received value, processes it (for example performs a scaling) and forwards the value to the secure data en-/decryption unit. This unit encrypts the data packet and forwards it to the bus medium. On the other end of the transmission channel, the secure data en-/decryption unit of the receiver decrypts the packet and forwards it to the actuator. It is obvious that the sender and the receiver must use the same unit.

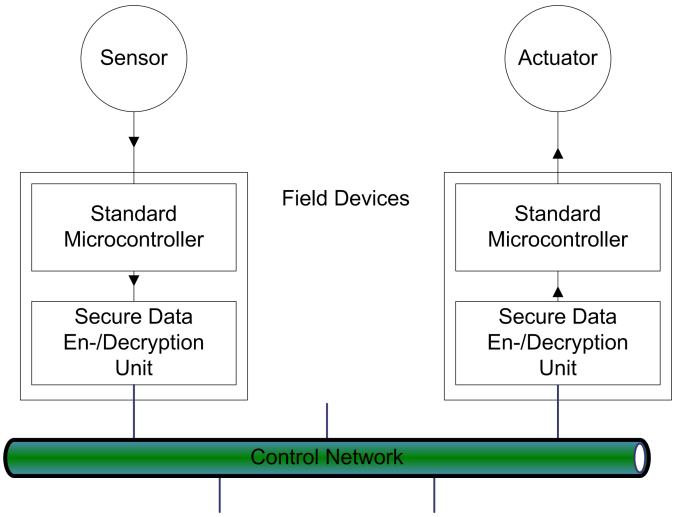


Figure 4.7: Secure data transmission with En-/Decryption Unit

There are several ways how such a secure data unit can be realised. An example is given in [2], where a smart card is used to en-/decrypt the data packets.

The solution mentioned above has one drawback. As encryption is performed after the microcontroller processes the data, a plaintext version of the measured sensor values is stored in the memory of the microcontroller. This means that if an attacker is able to read out the content of the memory (for example through the management system), it is possible to obtain a plaintext version of the sensor values.

A possible solution is illustrated in [10]. Using this solution, the data packets

are encrypted before they are read by the microcontroller. This means that the en-/decryption unit is placed between the sensor and the microcontroller. With this approach, the microcontroller never stores the plain version of the data. Unfortunately, there is still one problem which remains. Since only the encrypted form of the sensor value is available, the microcontroller cannot process it anymore (for example perform a scaling of the value).

Another possibility is to put the microcontroller and the en-/decryption unit into a safe containment. This containment must avoid an unauthorised physical access. As mentioned before, a seal or a sensor can be used to detect malicious damage. Additionally, the management system must be protected to avoid unauthorised usage. For example, it must be guaranteed that the content of the memory is only modified or read out by authorised entities.

4.2.2 Attacking the Control Network Medium

As mentioned at the beginning of this section, an attacker has two possibilities to attack a building automation system from the inside. In the previous section, it was shown how a single field device can be attacked and how such an attack can be avoided.

Another possibility is to attack the control network medium of the building automation system directly. To achieve this, the attacker must gain physical access to the control network medium. For example, an intruder could try to locate the physical network cable, intending to attach to the medium.

After the attacker has gained access to the medium, the attacker has four possibilities to disturb the network communication. In Section 3.1.1, these four different threats are described. Figure 4.8 illustrates these threats.

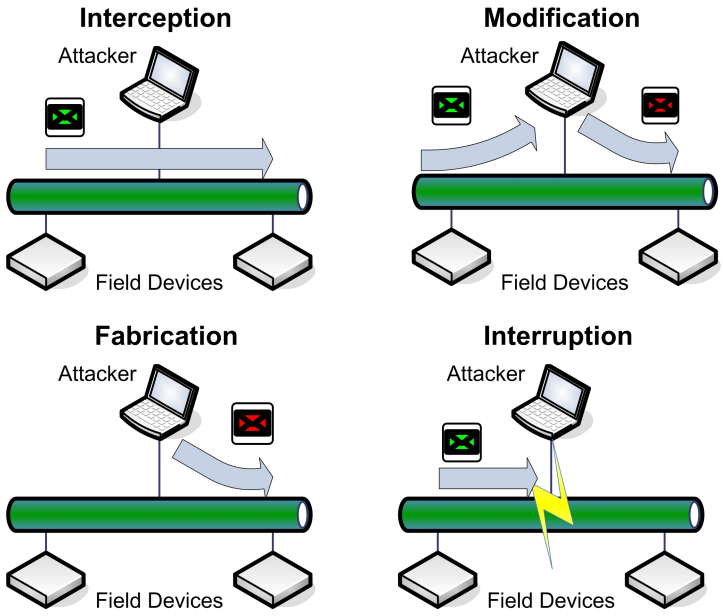


Figure 4.8: Possible Security Threats at the Control Level

To protect the control network against these security attacks, it is necessary to provide a secure channel. As mentioned in Section 3.2.1, such a secure channel avoids an unauthorised modification, fabrication and interception. To achieve such a protection, several security mechanisms are necessary. Another important issue is protection against interruption (for example Denial-of-Service attacks). Mechanisms which help to avoid and detect these attacks are discussed in the next Section 4.2.3.

As mentioned in the previous section, field devices contain low cost hardware components which have limited memory and limited computing power. Therefore, it is not possible to use standard techniques to protect the transmission channel of a control network. For example, a lot of these well-known security protocols (like TLS) use an asymmetric encryption scheme which is not suitable for small embedded devices. As shown in appendix A, a symmetric algorithm is much faster than an asymmetric one. Therefore, it is often necessary to use security mechanisms which are based on symmetric algorithms.

In [48], a security protocol called **SPINS** is described. This protocol was designed for the use in sensor networks which contain small, wireless sensors that run on batteries. The main objectives of SPINS are to provide data confidential-

ity, data authentication, data integrity and data freshness[4]. To guarantee that the protocol is suitable for small embedded devices, SPINS uses only symmetric algorithms. Additionally, the memory requirements are kept as small as possible. To satisfy these needs, the different cryptographic primitives (encrypted messages, hash values, random data,...) are generated out of one single algorithm. So, it is possible to implement the protocol stack of SPINS on devices that have limited memory capacity. The protocol itself works as follows.

SPINS assumes that the senor network consists of one or more base stations and several sensor nodes. The base station acts like a server and provides an interface to the outside network. The protocol itself consists of two building blocks: **SNEP** and μ**TESLA**. SNEP provides data confidentiality, authentication, integrity and freshness for point-to-point communication whereas μTESLA is responsible for providing authentication for broadcast communication.

As mentioned above, SNEP secures two-party communication. At the beginning, each sensor node retrieves an initial, secret master key. This master key χ_{AS} is shared between the node A and the base station S[5]. Since this master key is symmetric, the keys χ_{AS} and χ_{SA} are identical. The master key is used to calculate other secret keys which are needed by the protocol. These keys are:

- Encryption keys K_{AS} and K_{SA}:
  The keys K_{AS} and K_{SA} are used to encrypt the data packets. To derive these keys from the master key, a pseudo-random function F is used. These keys are calculated as follows: $K_{AS} = F_{\chi_{AS}}(1)$ and $K_{SA} = F_{\chi_{AS}}(3)$.
- MAC[6] Keys K'_{AS} and K'_{SA}:
  These two MAC keys are used to calculate the message digest of a data packet. Again, a pseudo-random function F is used to retrieve these keys. These keys are calculated as follows: $K'_{AS} = F_{\chi_{AS}}(2)$ and $K'_{SA} = F_{\chi_{AS}}(4)$.

To achieve **data confidentiality**, the messages are encrypted using the corresponding shared, secret key. This means that if an entity A wants to send confidential data to the base station S, it encrypts the data using the calculated key K_{AS}. Since S is also able to calculate the key K_{AS} (A and S share the same master key), S can decrypt the message. If a base station S wants to send confidential data to an entity A, it uses the secret key K_{SA} to encrypt the transmitted data.

This basic protection against interception is not enough. To avoid replay attacks, another mechanism is necessary. Therefore, SNEP uses a counter to achieve

[4]Data freshness ensures that an attacker is not able to replay an old message.

[5]The SNEP protocol assumes that a sensor node communicates only with its corresponding base station. A communication between two sensor nodes is not possible.

[6]Message Authentication Code

data freshness. This counter uniquely identifies each message. The counter guarantees that multiple encryptions of the same plaintext are always different. This means that the attacker is not able to replay previously sent messages . To achieve this, the counter size must be large enough to guarantee that each counter is unique during the whole life cycle of the system (for further details see 3.2.2).

A common approach is to transmit the counter value together with the data. The main drawback of this approach is that the total message length increases. This means that more energy is needed to transmit the message. Therefore, SNEP uses two counters which are shared by the two communication participants (one for each direction). After a message has been transmitted or received, both participants increment the corresponding counter. The counter itself is used as input parameter for the encryption algorithm. So, it is not necessary to transmit the counter together with the user data. The whole message is calculated as follows (C denotes the encrypted version of message P and $N_{A \to S}$ denotes the corresponding counter):

$$A \to S : C = E(K_{AS}, N_{A \to S}; P)$$
$$S \to A : C = E(K_{SA}, N_{S \to A}; P)$$

The mentioned counter mechanism has one drawback. If a message gets lost, the counters of the communication parties will get asynchronous. A possible solution is to implement a **counter exchange** protocol. If an entity recognises that the counters are not synchronous anymore, the entity can initiate a counter exchange. A detailed description of such a protocol can be found in [48].

To provide **data integrity** and **two-party authentication**, a MAC is used. To calculate this MAC, a one-way hash function is used (for further details see 3.2.3). For security reasons, SNEP uses a separate key to calculate the MAC. The MAC is calculated as follows (H denotes the used one-way hash function and $||$ denotes the concatenation function):

$$A \to S : M = H(K'_{AS}, N_{A \to S}||C)$$
$$S \to A : M = H(K'_{SA}, N_{S \to A}||C)$$

The encrypted message C and the MAC M are transmitted together to the receiver. Figure 4.9 summarises the different steps of SNEP.

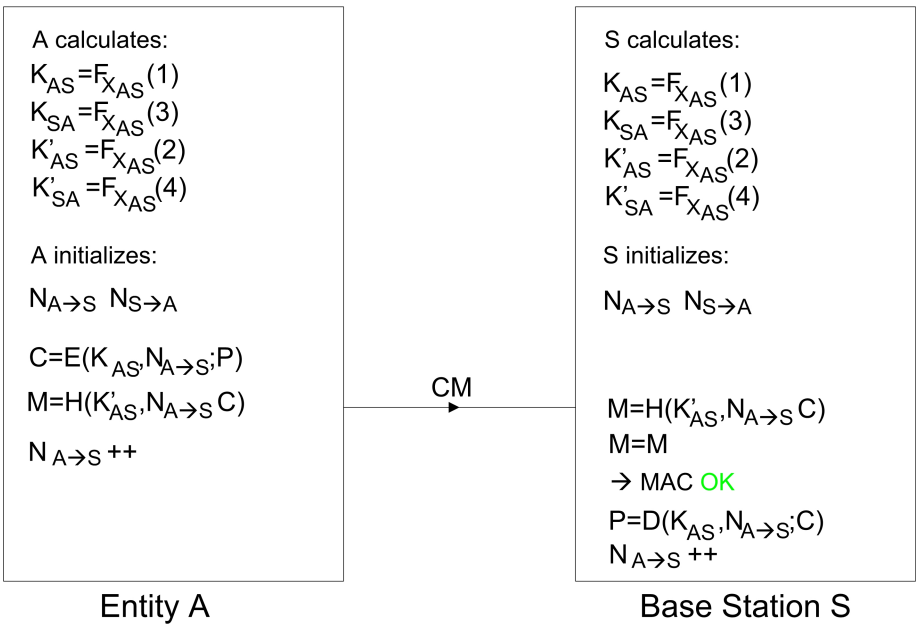


Figure 4.9: SNEP

As mentioned before, SNEP provides only a two-party authentication. In a building automation system, broadcast and multicast is often used as process data communication. Therefore, an authenticated broadcast protocol is necessary. Most available solutions are based on asymmetric algorithms (for example digital signatures). As asymmetric algorithms are too slow to run on embedded devices, another solution has to be found. Therefore, the second security building block called μ**TESLA** is responsible for providing an authenticated broadcast. μTESLA is similar to the standard TESLA protocol (for further details on TESLA see [65]). Compared to the standard TESLA protocol, μTESLA uses only symmetric cryptographic algorithms. Therefore, it can be run on small embedded devices (for a detailed description of the μTESLA protocol see [48]).

To save memory space, SPINS uses only one single block cipher algorithm to generate all necessary primitives. This single block cipher (in [48] RC5 was chosen) is used as pseudo-random generator, encryption/decryption function and MAC calculator.

To save additional memory space, the block cipher algorithm mentioned above is used for encryption and decryption. As mentioned above, SPINS was designed to be suitable for small devices that run on batteries. Sending and receiving of

messages consume a lot of power, especially if a wireless network medium is used. Therefore, it is essential to keep the message length as small as possible. As mentioned before, a counter is used to avoid replay attacks. To avoid an increase of the message length, the required counter value is not added to the content of the message. Figure 4.10 illustrates this concept. Suppose, an entity A wants to transmit a data packet to a base station S. First, a so called one time pad is generated. To get this pad, the current counter value $N_{A \rightarrow S}$ is encrypted using the shared secret key K_{AS}. This one time pad is XORed with the plaintext P. After S has received the cipher text C, S calculates the same one time pad. To get the plaintext out the received message C, C is XORed with the generated one time pad.

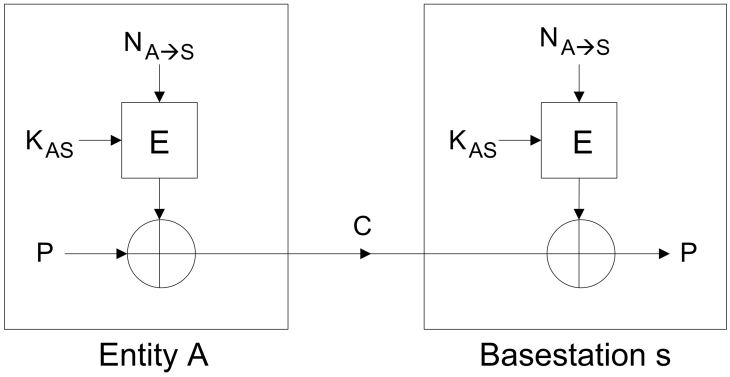


Figure 4.10: Encryption and decryption in SNEP

To calculate the MAC, the same cryptographic algorithm is used. Due to security reasons, a separate key K' is used to calculate the MAC. To get a MAC with a fixed length l, the following scheme is used. The first l bits of the input are encrypted using the secret key K'. Then, the result is XORed with the next l bits. These bits are encrypted once again. This mechanism is iterated until the whole message has been processed. Figure 4.11 illustrates this approach.

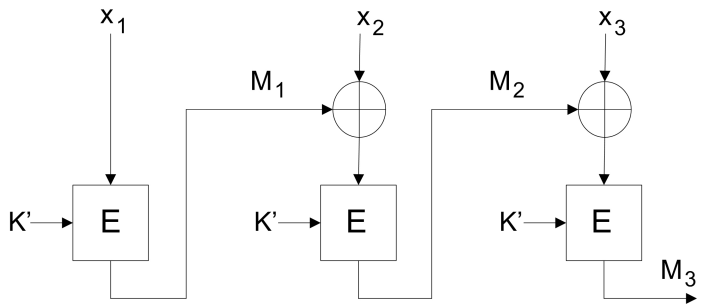


Figure 4.11: MAC generation in SNEP

As mentioned before, the protocol needs a pseudo-random function to calculate the necessary keys. To reduce code space, the MAC function is used to compute pseudo-random numbers (for further details see [48]).

4.2.3 Denial-of-Service Attacks

To disturb the network communication, an attacker could try to perform a so called denial-of-service (DoS) attack. A DoS attack has the objective to make a service or data unavailable. To achieve this, the attacker is trying to waste network and system resources in order to avoid the target from performing its expected function. For example, to interrupt the communication between two network nodes, an intruder could try to consume as much network bandwidth as possible. Another example is where an attacker tries to overflow the internal buffers of a network node. For example, if the attacker establishes too many connections, the internal buffer may overflow and the node is not able to handle incoming requests anymore.

As mentioned in the previous sections, additional security mechanisms are necessary to protect the system against DoS attacks. To achieve an effective protection, it is necessary to detect abnormal operations which may indicate a DoS attack. Therefore, an integrated intrusion detection system can help to discover such a possible DoS attempt. It is important to note that a detection is not always sufficient. If a DoS attack has been discovered, it could be too late to avoid damage. Therefore, it is necessary to implement mechanisms which protect the network against DoS attacks.

In [11], different kinds of DoS attacks are described. Each layer of the network architecture is vulnerable to other types of DoS attacks. Table 4.1 gives an

overview about these different attacks. Additionally, the corresponding network layer and examples of defence mechanisms are listed.

Layer	**Attack**	**Defence**
Physical	Jamming	Frequency Hopping
	Tampering	Safe Containment
Link	Collision	Error Correction Codes
	Exhaustion	Rate Limiting
	Unfairness	Small Frames
Network	Neglect and Greed	Redundancy
	Homing	Encryption
	Misdirection	Authentication
	Black-Hole	Authentication
Transport	Flooding	Client Puzzle
	Resynchronisation	Authentication

Table 4.1: Different kinds of DoS attacks

One of the most well-known attacks on wireless networks is called **jamming**[7]. Jamming means an attacker tries to interfere the radio frequencies that the wireless network is using to transmit the data packets. This jamming causes a disruption of the communication between the other network nodes. Frequency hopping can help to avoid this kind of attack. If the attacker is able to follow the hopping or a wide section of the frequency band is jammed, jamming is still effective. Another possible solution is to report the attack to the neighbour and to reroute the traffic. For a detail description of this solution see [11].

Another kind of attack is called **tampering.** Tampering means the attacker tries to attack the network node physically. As mentioned in Section 4.2.1, a safe containment can be used to detect such physical attacks.

At the link layer, an attacker could force a checksum mismatch. To achieve this, the attacker simply needs to introduce a **collision** in at least one octet of the transmitted data. This change causes a disruption of the whole data packet and therefore a checksum mismatch. Error-correction codes can help to detect and correct such a corruption. It is important to note that these error-correction codes can only correct a small amount of errors. This means that if the attacker disrupts

[7]It is important to note that jamming can also be performed on wired media. For example after an attacker has gained physical access to the network medium, he could try to send a disturbing signal.

more data than the code can correct, a corruption of the data packet cannot be avoided. A full defence against such attacks is often not possible.

Another type of attack is also known as **exhaustion**. If a CSMA/CD protocol is used, an attacker could attempt to generate a collision. Such a collision causes a retransmission of the whole data frame. If the network node runs on batteries, these retransmissions could result in the exhaustion of the battery. To avoid this, time-division multiplexing can be used. Another possible solution is to introduce rate limiting (for further details on rate limiting see [11]).

Unfairness is another type of DoS attack. The attacker uses the priority mechanisms to transmit high-priority messages. This means that low priority messages have to wait until the transmission of higher ones have been finished (starvation). A possible solution is to use small frames. Using small frames means the attacker can reserve the network channel only for a short time.

At the network layer, an attacker can use the routing mechanisms to disturb the communication. Such a DoS attack that affects the routing of a network is called **neglect and greed**. Instead of routing the data packets, an attacked network node randomly drops the data packets. Additionally, the malicious node can still acknowledge the dropped packets. Especially, networks that use dynamic source routing are vulnerable to this kind of attacks. To avoid such attacks, multiple routing paths or sending redundant messages can be used.

In a building automation system, some nodes implement special functionality. These nodes provide services that are needed by other entities (for example routers, key servers, ...). To interrupt the communication, an attacker could try to attack these vital devices. An attack which has the main aim to make a vital service unavailable is called **homing**. To avoid homing attacks, it is essential to protect these special devices against malicious attacks. To find the location of such devices, the attacker has to observe the network traffic. Based on this network traffic, the attacker can detect the presence of such a device. To hide these important nodes, the attacker must not be able to analyse the network traffic. To achieve this, the data packets must be encrypted. It is important to note that the headers of the packets must be encrypted too. Otherwise, the attacker is able to get relevant information (for example, source and destination address, routing information,...).

Another possibility is to route the message along a wrong path (**misdirection**). To achieve this, an attacker could fabricate additional routing information. Again, network protocols that use dynamic source routing are particularly vulnerable to such attacks. To avoid an adding of additional routing information, it must be guaranteed that only authorised users are allowed to change routing information.

Therefore, an authentication mechanisms could be used to protect routers and gateways against unauthorised access.

In distance-vector based network protocols, so called **black holes** can be used to disturb the routing of the network traffic. If a malicious node pretends zero-cost routes, other nodes will route their traffic in the direction of this node. Since the network traffic is routed to this malicious nodes, this point is called a black hole. Again, authentication can be used to avoid such black holes.

Another well-know DoS attack is called **flooding**. In this case, the attacker tries to overflow the internal buffers of the target node. For example, an attacker could establish many connections to a node. For each incoming connection request, the node must allocate resources to store necessary information about this open connection. If there are too many open connections, the memory will get insufficient. So, the node is not able to handle connection requests anymore.

To avoid such flooding attacks, a so called **client puzzle** ([49]) can be used. A client puzzle makes a request for the client at least as expensive (in terms of computational costs) as for the server. This scheme works as follows. If a client wants to establish a connection, the client sends a request to server. If the server is busy, the server sends back a puzzle which the client has to solve. After the client has solved the puzzle, it sends the solution to the server. The server verifies the solution and if the solution is correct, the server accepts further requests. To achieve an effective protection against DoS attacks, the effort to solve the puzzle must be higher than the effort to verify the solution. For example, in [50], a hash value is used as puzzle. The client has the aim to find the input value which produces this hash value. To achieve this, the client has to solve this problem by brute force. As it is very easy to verify whether the solution is valid or not, the client must pay more computing costs than the server (for further details see [49]).

It is obvious that the complexity of the client puzzle depends on the computing power of the involved network nodes. As mentioned in Section 2.5, in a building automation system, field class devices have limited computing power. If a client puzzle is used to avoid DoS attacks, it is important to note that an attacker could use a high performance device (for example a management device or a PDA) to perform a DoS attack. This means that the puzzle must be difficult enough to guarantee that it takes more time to solve the puzzle than the server needs time to verify the result. It is important to note that it must still be possible that other field devices are able to solve the puzzle in a reasonable time. If the attacker has much more computing power than a field class device, it is difficult to define a client puzzle which satisfy these mentioned needs. Therefore, a client puzzle is always not suitable to avoid DoS attacks.

In the header of the transport layer, details about the current state of the connection is stored (for example sequence numbers, control flags, routing information,...). An attacker could take advantage of these header information. The attacker could attempt to send invalid messages which cause a **desynchronisation** of these values. For example, an attacker could send messages with invalid sequence numbers to force the communication participants to perform a resynchronisation. Again, authentication can be used to solve this problem.

Sometimes it is not possible to achieve a full protection against DoS attacks. Anyhow, to reduce the resulting damage, it is necessary to isolate the affected part of the network. To achieve this, a defence-in-depth approach can be used. If a DoS attack is detected, the affected network segment must be separated from the rest of the network. If the malicious nodes can be isolated, the rest of the network can be kept operable. To achieve this, it must be possible to decouple the corresponding network segment. For example, a gateway or router can cut off the affected network segment and reroute the network traffic.

As mentioned before, security has to be considered right at the beginning of the development. Therefore, the software design itself must take security considerations into account. It must be ensured that the security of a system can always be guaranteed, even under a DoS attack. This means that the system must change in a failsafe state. Consider, for example, a security alarm system that consists of several sensor (see Figure 4.12). These sensors are responsible for detecting unauthorised intrusion (for example presence detectors or glass break sensors,...). A possible, naive solution is shown in the left part of Figure 4.12. If a sensor detects a possible intrusion (for example a glass break sensor detects a broken window), the sensor sends a message to the alarm annunciator. As long as the delivery of the message can be guaranteed, this scheme works fine. If a DoS attack interrupts the communication channel between the sensor and the alarm annunciator, the message cannot be delivered anymore. So, the intrusion cannot be detected. A more secure approach is to periodically send the current state of the sensor. If the alarm annunciator does not receive any messages from the sensor, it knows that something is going wrong. To be on the safe side, the alarm annunciator could signal this abnormal behaviour.

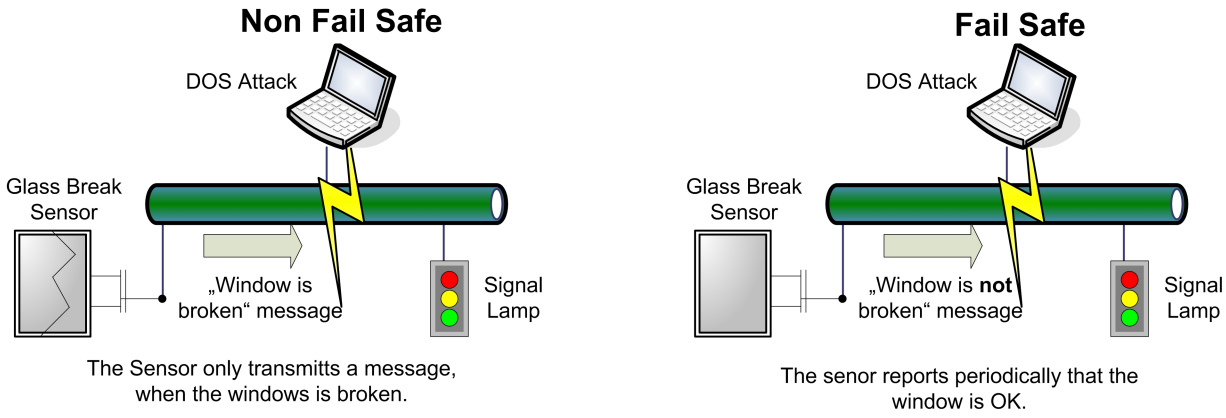


Figure 4.12: Non-Fail-Safe vs. Fail-Safe Scheme

As mentioned in this section, DoS attacks are hard to handle. Especially in control networks that consist of field devices with limited system resources, protection against DoS is not a trivial task. Therefore, it is essential to consider security rights at the beginning of development.

Chapter 5

Security Concepts in Building Automation Solutions

In the previous section, it was shown how a building automation system can be attacked. Different security threats and their importance were analysed. To avoid malicious attacks, several security mechanisms must be implemented. Therefore, possible solutions and concepts that help to discover or avoid security attacks were presented.

In this chapter, a closer look at available building automation solutions is taken. This chapter gives an overview about existing solutions and their implemented security mechanisms. After providing a brief introduction into these solutions, the implemented security concepts will be analysed.

Therefore, the next Section 5.1 includes a description of Echelons LonWorks. In the following Section 5.2, the security features of BACnet will be analysed. Finally, the last section of this chapter will take a closer look on the EIB/KNX standard.

5.1 LonWorks

LonWorks was developed by Echelon Corp. It consists of the LonTalk communication protocol, a controller (Neuron Chip) and a management tool. In 1999, LonTalk has become the formal standard ANSI/EIA-709 (for further details see [54]).

LonTalk supports different network media. It is possible to use twisted pair, powerline and fibre optic as network medium. Wireless solutions (radio fre-

quency) are also available but no standard profile for these wireless media exist yet.

The most common network medium is twisted pair. Depending on the used topology and necessary bandwidth, different twisted pair profiles exist. The most popular profile (FT 10) allows a maximum bandwidth of 78.1 KBits/s. It allows free topology and a maximum cable length of 500m. In addition to FT10, higher-speed profiles (for example TP-1250 with a bandwidth of 1.25 MBits/s) are also available. All twisted-pair media use a predictive p-persistent CSMA scheme. Compared to CSMA/CD protocols (for example Ethernet), this scheme ensures a minimum data rate even under heavy load.

In a LonWorks system, it is also possible to use an IP based network as backbone. To achieve this, **LonWorks/IP** has been defined which is part of the standard ANSI/EIA-852 (for further details see [55]).

A LonWorks network can be divided into different domains. Each domain has a unique domain ID which is up to 48 bits long. To reduce the message length, this ID can be shorter than 48 bits. Each domain can be divided into up to 255 different subnets. Each subnet can contain up to 127 LonWorks nodes. The different domains are interconnected via so called domain gateways. Due to the design of the address space in LonWorks, such a domain gateway operates at the application layer. It is responsible for transferring the data between the different domains. It is important to note that these devices do not provide a protocol conversion. Strictly speaking, a domain gateway act as a proxy and therefore the term gateway is not appropriate. To interconnect the different subnets, routers are used. These routers are responsible for routing the network traffic between the different subnets.

Each domain can contain up to 256 multicast groups. The members of a particular group do not need to be in the same subnet. They can be located in different ones. Generally, the LonTalk protocol supports unicast, multicast and broadcast (both subnet and domain broadcast) communication. In addition to an unacknowledged mode, a reliable, acknowledged unicast and multicast transmission mode is also available. It is important to note that if acknowledged multicast is used, a domain can only contain 64 different groups.

In addition to this logical addressing scheme, each node has a world-wide unique node ID. This node ID is 48 bits long and can be used by management services. Normal data transmission uses the logical addressing scheme mentioned above.

In LonTalk, distributed applications typically communicate using **network variables**. Such a network variable is an abstract object which represents a data value in the network. A network variable may be connected to multiple devices.

The main benefit of such a network variable is that whenever the value of the network variable is changed locally within the node software, the system software automatically creates and transmits a data packet. On the other hand, if the device receives a data message, the value of the appropriate network variable is automatically updated (for further details see [51]).

For further details on LonWorks and LonTalk see [1] and the mentioned standards ANSI/EIA-709 ([54]) and ANSI/EIA-852 ([55]).

5.1.1 Security Concepts

It is important to note that LonTalk only provides an authentication service. Data encryption is not supported. Since all messages are transmitted in plaintext, data confidentiality cannot be guaranteed.

The integrated authentication mechanism is implemented at the transport and session layer of the protocol. It is a four step challenge-response mechanism which provides a basic form of data authentication. This mechanism works as follows:

Suppose, an entity A wants to send an authenticated message M to an entity B. To start the challenge-response mechanism, A sets the authentication bit of the message M and sends it to the receiver. After B has received the message, B generates a 64 bit random number N. This random number N is sent back to the sender A. Immediately after B has transmitted the random number, B takes this random number N together with the received message M and calculates a 64 bit hash value using a one-way hash function H and an authentication key K (48 bit long). After having received the random number N, A also computes this 64 bit hash value using the same authentication key K. This 64 bit hash value is sent back to B. To verify the authentication of the message, B compares the calculated hash value with the received one. If the two hash values are identical, the authenticity has been proven. Figure 5.1 illustrates this challenge-response mechanism.

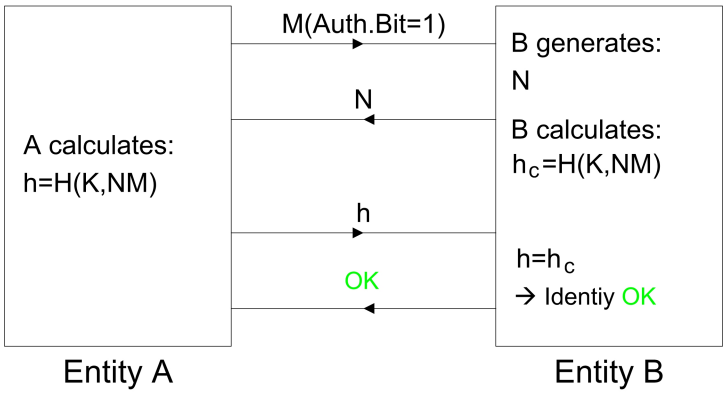


Figure 5.1: Authentication in LonTalk

To achieve this authentication mechanism, LonTalk provides different authentication service primitives (for further details see [54]). It is important to note that the received message is always passed together with a notification to the application layer. This notification informs the application layer whether the authentication was successful or not. This means that the application itself is responsible for processing the result of the authentication process.

If network variables are used and authentication is required, both communication participants must activate authentication in the description of the network variable. Additionally, the sender of the value of the network variable must set the authentication bit to initiate the challenge response protocol.

It is obvious that this protocol provides only a basic form of authentication. One drawback of the LonTalk authentication protocol is that the authentication bit of the first message must be set to initiate the challenge-response protocol. This means that the receiver has no opportunity to force the sender to prove its identity.

Another problem is that only the sender is able to prove its identity. With this authentication scheme, it is not possible to verify the identity of the receiver. So, the sender cannot be sure that the receiver is what it claims to be.

The usage of the authentication protocol is restricted to acknowledged unicast and multicast. This means that if an unacknowledged transmission mode or broadcast is used, the identity of the sender cannot be verified.

As shown in Figure 5.1, four messages are needed to complete the authentication process. Another drawback of this scheme is that these four messages are always necessary to perform authentication. Even if a sender transmits multiple data packets to the same receiver, the challenge-response mechanism must be per-

formed for each data packet. It is not possible to establish a session which requires only performing the challenge-response protocol at the beginning of the session.

Another problem with authenticated multicast is that each receiver generates its own random number and sends the random number to the sender. To prove its identity, the sender must respond to all receivers with the corresponding hash value. This means that if a multicast group contains n members, the sender must calculate $n - 1$ hash values.

The authentication protocol is vulnerable to DoS attacks. As mentioned before, the receiver computes the hash value immediately after the random number is calculated. To start a flooding attack (for further details see 4.2.3), the attacker sends a lot of messages with a set authentication bit. For each message, the receiver generates a random number and calculates the necessary hash value. As it is time-consuming to calculate these hash values, these incoming messages will result in a DoS attack.

Another security flaw is the cryptographic algorithm itself. Since the algorithm is not publicly available, it cannot be verified whether the algorithm is secure or not. The algorithm uses 48 bit keys which is too short to be secure enough. Therefore, the used cryptographic algorithm can be thought as being weak.

Another problem is the key distribution. The LonTalk protocol does not provide a mechanism to distribute the secret keys in a secure manner. Since the keys must be transmitted in plaintext, the initial key distribution has to be done in a secure environment to ensure that nobody intercepts the keys. A possible solution is to connect the device directly to the management node.

The authentication protocol has another drawback. It is only possible to store one authentication key. This means that all entities that want to communicate with each other must share the same authentication key. To use authenticated multicast, all members of the group must have the same key. But it is important to note that the identity of the sender cannot be verified exactly. The receiver can only ensure that an owner of the key has calculated the hash value. Therefore, the receiver only knows that a member of the group has computed the hash value.

5.2 BACnet

In 1987, the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) project committee began with the development of BACnet (Building Automation and Control networking protocol). The main objective was to provide a solution for building automation systems of all sizes and types.

In 1995, the development has been finished and BACnet was published as an ANSI/ASHRAE standard. Later in 2003, BACnet has become a CEN and ISO standard ([53]). Since the first release, the BACnet specification is under further development.

BACnet does not define the underlying network layers (layer 1 and 2). To achieve compatibility, BACnet supports five different network technologies which have been standardised. These five network types are:

- Ethernet
- ARCNET
- Master-Slave/Token-Passing (MS/TP)
- LonTalk
- Point-to-Point (PTP)

As shown above, LonTalk is also supported. It is important to note that only BACnet specific messages can be used.

In 1999 **BACnet/IP** was introduced. Basically, two mechanisms exist to use BACnet over IP networks. First, it is possible to use a tunneling approach. A special device called B/IP PAD[1] encapsulates the BACnet message into a UDP packet. This packet is transmitted to the destination B/IP PAD where it is reconverted into a BACnet message.

Another possibility is to use UDP directly as link layer protocol. To achieve this, BACnet Virtual Link Layer (BVLL) was defined which provides the opportunity to use UDP as data link layer protocol. To support broadcast communication, a special device called BACnet Broadcast Management Device (BBMD) is required.

In a BACnet system, a physical network line is called a segment. To extend the maximum cable length, repeaters and bridges can be used to link different segments. An interconnection of such segments is called a network. These BACnet networks can linked together to form an internetwork. To achieve this interconnection, BACnet routers are used. These routers are responsible for routing the data packets between the different networks.

Each BACnet network has a unique 2 byte BACnet network number. To identify the node in a network, each network member has a local address which is up to 255 bytes long. This local address is used by the link layer of the network. Therefore, the exact length of the local address depends on the used network medium. For example, if Ethernet is chosen as network medium, the MAC address is used as local address.

[1] BACnet/Internet Protocol Packet-Assembler-Disassembler

In BACnet, the used data primitives are called objects (**BACnet objects**). An object is a collection of data elements which forms a particular function of the system. Such an object has a set of properties. Each property represents a definite data value. For example, a "temperature object" can consist of a property `current_value` and other properties like `min_value`, `max_value` and `resolution`.

The BACnet standard currently defines 25 different object types. Each node can have several objects. The `Device` object which contains general information about the node must be present in each node. As mentioned above, each object can have several properties. In the BACnet standard, nearly 200 different properties are defined. Three of these properties must be present in each object. These three standard properties are `object-identifier`, `object-name` and `object-type`.

To perform operations on these objects, BACnet provides different services (**BACnet services**). The BACnet standard currently defines 40 different application services which are divided into five different categories. These five categories are:

- Alarm and Event
- File Access
- Object Access
- Remote Device Management
- Virtual Terminal

To access and manipulate BACnet objects, the services `ReadProperty`, `WriteProperty`, `ReadPropertyMultiple` and `WritePropertyMultiple` can be used. From these services, only the `ReadProperty` service must be implemented in each device.

BACnet uses a client/server model. To handle events, BACnet provides three different handling methods. These three methods are "Intrinsic Reporting", "Algorithmic Change Reporting" and "Change of Value". For further details on BACnet see [1] and [52].

5.2.1 Security Concepts

The BACnet protocol provides limited security mechanisms that guarantee data confidentiality and data integrity. Additionally, authentication mechanisms are supported which can be used to verify the identity of peer entities, data origin and operators. The necessary mechanisms are defined in clause 24 in the BACnet standard (see [52]).

To achieve the mentioned security objectives, a cryptographic algorithm and an authentication mechanism are used. To encrypt data packets, BACnet uses the symmetric DES algorithm. The authentication mechanism is based on a trusted keyserver which is responsible for generating and distributing session keys. These session keys are used to encrypt the transmitted data between two entities. To establish a secure connection to the keyserver, each node must have a secret key. The distribution of these private keys[2] is not defined in the BACnet standard.

Figure 5.2 illustrates the basic authentication mechanism. To obtain a session key, A sends the device identifiers A and B to the keyserver S. This is done by using the `RequestKey` service. To protect the request, A encrypts the message with its private key K_{AS} which is shared between the entity A and keyserver S. After S has received the request, it generates a 56 bit session key K_{AB}. Then, S encrypts the session key together with the BACnet address of B using the private key K_{AS}. This encrypted message is sent to A using the service `AddListElement`. This service adds the session key to the `List_Of_Session_Keys` property of the `Device` object of entity A. Additionally, the session key is also transmitted to B. To achieve this, S sends the encrypted version of K_{AB} and the BACnet address of A to B.

After both communication participants have received the session key, message encryption or authentication can be performed. To perform peer authentication, the `Authentication` service can be used. Peer authentication is shown in Figure 5.2 and works as follows:

Entity A generates a random number N which is encrypted using the session key K_{AB}. Then, A sends this encrypted random number to entity B. B decrypts it and performs an inversion of the most and least significant bit of N. Then B encrypts this transformed random number N' using the session key K_{AB} and sends it back to A. To verify the identity of B, A decrypts the received message and tests whether the modification is valid. Figure 5.2 illustrates this scheme.

[2]It is important to note that in BACnet, a private key is a symmetric, secret key. In this context, a private key is not the same as an asymmetric private key which is used in a public key system.

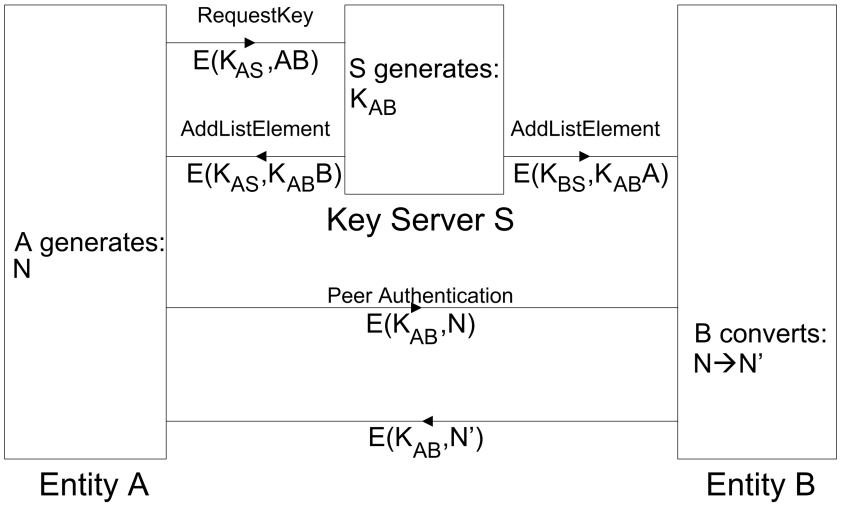


Figure 5.2: Authentication in BACnet

Clause 24 of the BACnet standard defines different security mechanisms. These mechanisms are:

- Peer Authentication
- Message Initiation Authentication
- Message Execution Authentication
- Operator Authentication
- Data Confidentiality
- Enciphered Session

Peer authentication uses the authentication service explained above. It is used to verify the identity of a BACnet node (for further details see Figure 5.2).

Message Initiation Authentication is used to verify the identity of the initiator of a sent message. To achieve this, the authentication service is extended. In the authentication request, the `Expected Invoke ID` parameter specifies which request should be authenticated.

The **Message Execution Authentication** service is similar to the `Message Initiation Authentication` service mentioned above. This service has the main aim to verify the identity of the server which should execute the requested operation. Again, the `"Expected Invoke ID` parameter is used to identify the request which should be authenticated.

The **Operator Authentication** service can be used to verify the identity of an operator. To log onto a particular device, the authentication service is extended.

In the authentication request, the parameters `operator name` and `password` must be used to specify the user name and the password of the operator. There are two possibilities to verify the password. First, the verification of the operator name and password can be done locally. If the entity has not the capability to verify the name and password locally, the keyserver can be used to validate the password. To achieve this, the authenticate request can be forwarded to the keyserver. After the keyserver has verified the password, the result of the verification is sent back.

To send **confidential data**, the communication participants must receive a session key from the keyserver. With this session key, it is possible to encrypt the data packets. Figure 5.2 illustrates this mechanism.

The BACnet protocol provides a mechanism to start and to end an **enciphered session**. Again, the authentication service is used. To start an enciphered session, the `Start Enciphered Session` parameter of the authentication request must be set to true. To verify the identity of the initiator, the message initiation authentication service should be used. After the initiator has proven its identity, the enciphered session can be started. To end the session, the `Start Enciphered Session` parameter must be set to false. Again, the message initiation authentication service should be used to verify the identity of the initiator of the request.

The main drawback of the security mechanisms mentioned above is that only DES is supported. Since DES uses short keys (56 bit), brute force attacks can be used to find valid keys. Therefore, DES is not secure and AES should be used instead (for further details see Section 3.2.1).

Another problem is the initial key distribution. The BACnet standard does not specify the generation and distribution of these private keys. In the BACnet standard, these mechanisms are considered as ”local matters”.

The implementation of the keyserver is also not defined by the BACnet standard. The exact implementation is left open. The keyserver holds a copy of all private keys. Therefore, it is obvious that the keyserver must be protected against all kinds of malicious attacks.

The authentication protocol itself has several security flaws. As mentioned in [56] and [57], the authentication mechanism is vulnerable to the following security attacks[3]:

- Man-in-the-Middle Attacks
- Type Flaws[4]

[3]Unfortunately, in [57] the exact methods how these attacks are performed are not discussed in detail.

[4]To cause a misinterpretation of the content of a message, the attacker tries to change a well-known sequence of bits. For example, keys and nonces are such typical well-known sequences.

- Parallel Interleaving Attacks[5]
- Replay Attacks
- Implementation Dependent Flaws

One problem of the mechanism is that the freshness of the session keys cannot be guaranteed. As mentioned before, each session key is added to the `List_Of_Session_Keys` property of the `Device` object. The BACnet specification does not define a finite lifetime of these session keys. Again, it is a "local matter". This means that if the freshness of the session keys is not verified, an attacker can use an old session key to communicate with a particular device. Therefore, it is essential to implement a mechanism to limit the lifetime of the session keys.

To solve the problem mentioned above, it is essential to use a stronger cryptographic algorithm (for example AES). To increase the security of the authentication mechanism, the protocol must be improved. For example, Kerberos can be used as a replacement of the standard BACnet keyserver (for further details see [56] and [57]).

5.3 EIB/KNX

The European Installation Bus (EIB) was developed as a field bus for installations in homes and building. Until 2002, the EIB specification was maintained by the EIB Association. In 2002, EIB was merged with Batibus and EHS (European Home System) and the KNX standard (see [58]) was defined. Additionally, the Konnex Association was formed which is responsible for the maintenance of the KNX specification. Since the definition of the KNX standard, EIB is also referred to as EIB/KNX.

EIB/KNX supports different network media. It supports the use of twisted-pair (TP) and powerline as well as a wireless solution (radio frequency). The most common type is TP1 which allows a maximum bandwidth of 9.6 KBits/s and free topology.

An IP tunneling mechanism called **EIBnet/IP** is also available. This mechanism provides the opportunity to use an IP based medium as backbone network. To achieve an encapsulation of the messages, IP tunneling routers are necessary.

To extend the maximum cable length of a physical segment, line repeaters can be used. An interconnection of one or more segments is called a line. A line can contain up to 254 devices. Up to 15 lines can be connected by a main line. Such

[5]This attack uses messages from one session in another concurrent session. An example of such an attack is shown in Section 3.3 (see Figure 3.8).

an interconnection of lines is called a zone. Up to 15 zones can be linked by a backbone line. To perform the interconnection, couplers are needed (line couplers and backbone couplers).

Each device has an individual address which is 2 bytes long. The first four bits of this address contains the zone ID and the second four bits the line ID. The last eight bits (the second byte) denotes the node ID which is used to identify the node in a line.

This individual address is used for unicast communication. Unicast communication is mainly used to transmit management messages. A connection-oriented, reliable form is also available. Additionally, multicast communication is also supported. With the multicast service, it is possible to send a message to a group of nodes. To identify the group, each group has an individual group address (2 bytes long).

This **group communication** mechanism is very efficient. Each node has a list of group addresses which the node belongs to (group address table). If a multicast message is received, the node tests whether the group address table contains the address of the received message. If the address is listed in the table, the incoming multicast message is processed. Otherwise, the node will ignore it. A node can subscribe to any group. It is important to note that the other group members as well as the sender of a group message do not need to know which nodes exactly belong to the group. If a node wants to join a group, it simply adds the corresponding group address to its group address table.

If a group message is sent, each member of the group acknowledges the reception of the message. Due to the bit coding of the acknowledgment frame, a negative acknowledgment always dominates on the bus. This means that a negative acknowledgment will always override a positive one. Therefore, all group members can send the acknowledgment frame at the same time.

EIB/KNX uses a shared variable model. The application related variables which are visible via the network are called **group objects**. Such a group object can be readable, writable or both. For example, a group object can be the state of a light switch (readable) or the output value of a relay (writable).

Each group object can belong to multiple groups. The group communication mechanism mentioned above is used to send or receive the data values of a group object. If a data value of a group object changes, the node sends a corresponding group message. To receive updates of a group object, it simply adds the corresponding group address to its group address table. In addition to this event driven approach, a polling mechanism is also available.

The most important standard component in an EIB/KNX system is the **Bus**

Coupling Unit (BCU). A BCU consists of a microcontroller (MC68HC05 family) and a transceiver which is responsible for the bus access. The system software of the BCU implements the network stack. Additionally, it provides an application environment to handle simple user applications.

Each BCU provides a local interface which is called **Physical External Interface (PEI)**. Via this interface, it is possible to attach application modules. Such an application module can be a simple light switch or a RS232 interface as well as an additional embedded device. To communicate via this interface, several protocols exist. For a detailed description of these different protocols see [59].

To maintain and configure an EIB/KNX system, a management tool called **EIB Tool Software (ETS)** is available. With this tool, it is possible to perform management tasks (for example assigning node addresses) as well as uploading user applications.

For additional information about EIB/KNX see [1] and [60] as well as [39].

5.3.1 Security Concepts

EIB/KNX does not support mechanisms that guarantee data confidentiality, data integrity, data authentication and data freshness. It provides only a basic access control scheme which is based on cleartext passwords.

This basic **access protection** mechanism can be used to avoid an unauthorised use of management procedures. To achieve this, 255 different access levels can be defined. Each of these access levels have different privileges. Access level 0 has the most privileges and access level 255 the least ones. For each of these access levels, a 4 byte password called key can be specified.

It is important to note that this access protection mechanism is only available on the BCU 2 (with a mask version of 2.0 or higher). Older BCUs (mask version < 2.0) do not support access protection. Additionally, current implementations of the BCU 2 system software only supports four access levels. The access protection mechanism of the BCU 2 can be used to protect the global access to a BCU 2 as well as the access to the memory and to EIB objects.

To restrict the **global access** to a BCU 2, four different access levels are available. Table 5.1 summarises these different levels.

Access Level	Usage
0	loading application programs
1	loading address table and association table
2	loading application parameters
3	access level without or failed authorisation

Table 5.1: Different Access Levels in EIB/KNX

Memory allocation is done by defining **memory control blocks (MCB)**. In these MCBs, a required read and write access level can be specified (4 bit write access level and 4 bit read access level). The mandatory access levels can be used to restrict the read and write access of the specified memory area. If a remote user wants to access the content of the memory, the system software checks whether the current access level is lower or equal than the required access level.

The read and write **access to EIB objects** can also be restricted. To achieve such a protection, it is possible to specify a required read and write access level. This can be done for each object separately. Again, if a remote host wants to access an EIB object, the system software checks whether the current access level is sufficient to perform the requested operation.

To gain access to a remote device, the following steps are necessary. Consider, for example, an entity A wants to connect to a remote device B to perform certain management tasks. To open a connection, A must send an `A_Connect` request. After the connection has been established, A sends an `A_Authorize_Request` message. This request contains the key for the required access level. B searches its list of keys and determines the access level for which the received key is valid. Afterwards, B stores this access level as the current access level. This access level is valid during the whole session. Then, B sends back an `A_Authorize_Response` message which contains this current access level. After A has received this response, it is possible to perform further operations (for example an `A_Memory_Read` request). After A has finished its tasks, it sends an `A_Disconnect` message to B which closes the connection. This means that B sets its current access level to the level which has the lowest access rights (if a BCU 2 is used, the current access level is set to 3). Figure 5.3 illustrates this example.

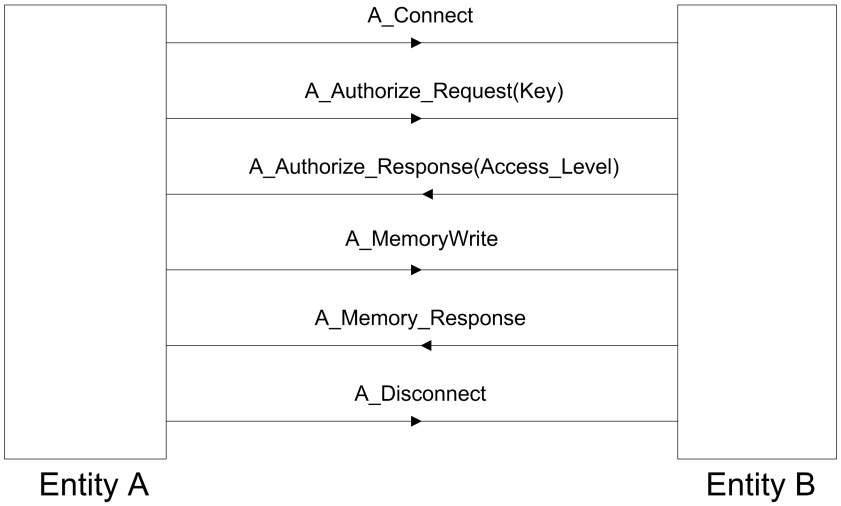


Figure 5.3: Access Protection in EIB

To change the key of a specified access level, the `A_Key_Write` service can be used. It is important to note that the current access level must be lower or equal than the access level which is specified in the request. Therefore, it is necessary to perform an authorisation request before the new key can be set. This means that the remote host must establish a connection (with the `A_Connect` request) and then an `A_Authorize_Request` message must be sent. To delete a key, the value 0xFFFFFFFF must be set as new key. This value is reserved for the empty password.

It is obvious that the mechanism mentioned above provides only a basic access protection. This access control mechanism has a lot of security flaws. One of the biggest problem is that the keys are transmitted in cleartext. Therefore, a key can be intercepted very easily (for further details see Section 6.1.2).

In Chapter 6, the security mechanisms of EIB will be analysed in detail. In addition to this analysis, different security flaws will be discussed. Additionally, at the end of Chapter 6, a possible solution will be presented.

5.4 Summary

The security mechanisms of LonWorks and EIB/KNX are not sufficient to fulfil the security requirements of a modern building automation system. As mentioned

in Chapter 4, a building automation system must be protected against different malicious attacks. It is important to note that the implemented security concepts of LonWorks and EIB/KNX cannot achieve an effective protection against these security threats.

As mentioned above, LonWorks supports data authentication. Compared to EIB/KNX, the integrated authentication mechanism provides authentication and integrity of data messages. It is important to note this mechanism is still to weak. Especially, the cryptographic algorithm seems to be insecure. Additionally, the protocol itself has security flaws and therefore further development is necessary.

EIB/KNX supports only an access protection mechanism which is only suitable to protect the system against a very limited class of malicious attack. Therefore, the rest of this thesis will take a closer look at security in EIB.

The security architecture in BACnet is more advanced. In addition to authentication and data integrity, an encryption of the transmitted data is also supported. Anyhow, the used cryptographic algorithm is obsolete and therefore it should be replaced by a modern one (for example AES). Additionally, the protocol itself must be improved to avoid certain security flaws.

A big problem which has not been solved by any of these three systems is the generation and distribution of initial secrets. Even if the architecture of the system itself is secure, a mechanism must be available which provides the opportunity to distribute the initial secrets (for example the shared secret key between node and keyserver) in a secure manner.

Table 5.2 gives an overview about the different security architectures. As shown in the previous sections, further development has to be done to provide a secure environment which is suitable for security critical applications.

System	**Authentication**	**Integrity**	**Confidentiality**	**Freshness**
LonWorks	64 bit MAC (48 bit key)	64 bit MAC (48 bit key)	-	Random number (64 bit)
BACnet	DES	DES	DES	Random Number
EIB/KNX	32 bit password	-	-	-

Table 5.2: Security Mechanisms in Building Automation Solutions

Chapter 6

Practical Experience: Security in EIB/KNX

As mentioned in the previous chapter, EIB/KNX provides only a basic access control mechanism. The implementation of this access protection is very rudimentary. An encryption mechanism that guarantees data confidentiality is missing as well as mechanisms that provide data integrity, authentication and data freshness. Therefore, the rest of this thesis will take a closer look at the security architecture of EIB/KNX.

The first part of this chapter examines the access control mechanism of EIB/KNX. An EIB test system was used to analyse the access protection in detail. Based on this analysis, different security flaws and problems will be discussed.

In the second part of this chapter, a possible solution of a secure EIB/KNX architecture is presented. Based on the practical experience of the analysis, security mechanisms are shown which fulfil the requirements of critical security applications.

6.1 Attacking password protection of BCU 2

As mentioned in Section 5.3, the security architecture of EIB/KNX is very rudimentary. The EIB/KNX protocol provides only an access control mechanism which is based on passwords. It is important to note that only BCU 2 (that means BCUs with a mask version higher or equal 2.0) supports this mechanism. Older BCUs do not provide this access control facility.

The main aim of this section is to analyse this access control mechanism.

To perform the necessary tests, an experimental EIB/KNX environment at the Automation Systems Group at the Technical University in Vienna was used. With this test system, the access protection of a BCU 2 was analysed and different vulnerabilities were determined. Figure 6.1 shows this test platform.

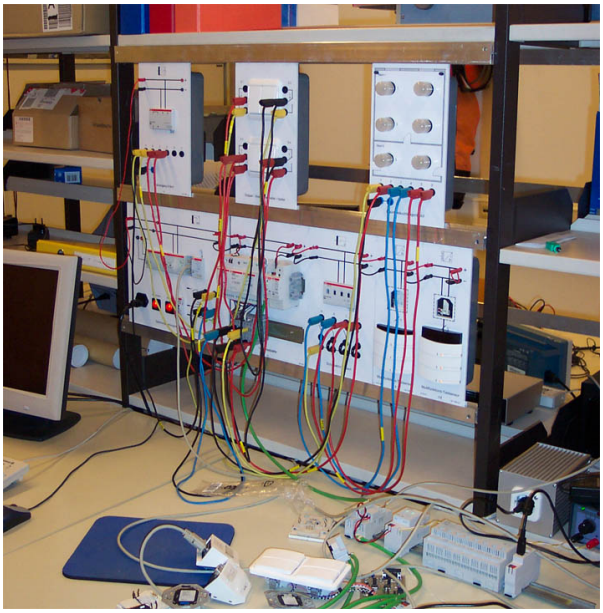


Figure 6.1: Experimental EIB/KNX environment

To analyse the protocol, two BCUs with mask version 2.0 were available (Siemens 5WG11142AB02 and Merten 6902 99). Additionally, the standard management tool ETS3 was also analysed. In addition to this Windows based tool, a Linux based EIB environment was available. This environment was also developed at the Automation Systems Group and consists of different Linux drivers (both kernel drivers and user mode divers; for further details see [67]) as well as miscellaneous EIB tools. With these drivers and tools, it is possible to access the EIB bus (via TPUART[1] or BCU). Additionally, a free development environment for BCUs was available. Using this BCU SDK, it is possible to develop user applications for BCU 1 and BCU 2 (for further details see [27] and [69]).

In addition to these available tools, an EIB/KNX testing tool was developed. With this tool, it is possible to observe the network traffic. Additionally, the tool

[1]Twisted Pair - Universal Asynchronous Receive Transmit

provides the opportunity to send EIB messages. This tool was written in Java and uses the Linux BCU 1 kernel driver to access the EIB/KNX network medium. Figure 6.2 shows a part of the user interface (for further details see [66]).

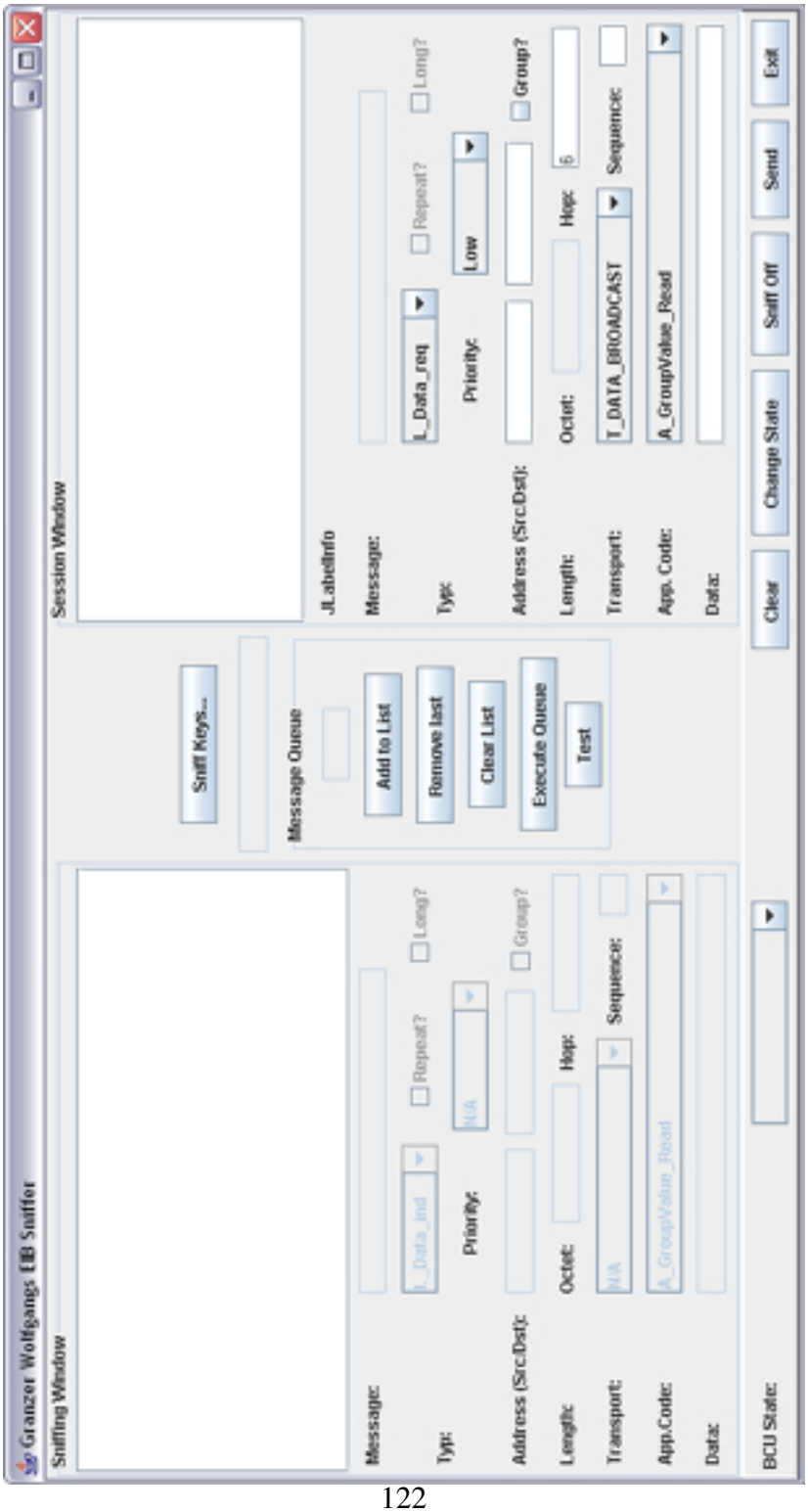


Figure 6.2: EIB Testing Tool

6.1.1 Brute Force Attack

Independent of the used system, an attacker has always the possibility to guess a password. To find a valid key, all different possible keys must be tested whether they are valid or not. This form of attack is called brute force attack.

In EIB/KNX, the access control mechanism uses 32 bit keys. This means that there are 2^{32} different keys which must be tested whether they are valid or not. Although, each BCU 2 could have a different set of keys, the management tool ETS uses a single key for the whole network. This means that the user can only choose one key which is used for all BCUs. Therefore, a brute force attack is appropriate to gain unauthorised access. If an attacker is able to guess one key, this key is valid for the whole network and therefore the attacker is able to gain access to any BCU.

To evaluate whether a brute force attack is suitable or not, the time it takes to try all possible keys must be calculated. To verify whether a key is valid or not, the following steps are necessary. First, the attacker must send an `A_Connect` message to establish a connection to the target of the attack. Afterwards, the attacker must send an `A_Authorize_Request` message which contains the key that should be tested. The target receives this authorisation request and responds with a transport layer acknowledgment (`T_ACK_PDU`). Then, the target sends an `A_Authorize_Response` back to the attacker. This response contains the access level for which the key is valid. To avoid a retransmission of the `A_Authorize_Response`, the attacker must also send back a `T_ACK_PDU`. Otherwise, the target will retransmit it.

To determine the time it takes to find a valid key, the transmission time of these five mentioned messages must be calculated. The maximum theoretical bandwidth of TP1 in EIB/KNX is $9.6 KBits/sec$. This means that the bit time is $104 \mu sec$. A single character called octet is transmitted as an UART character. TP1 uses a 8e1 UART encoding scheme. This means that it takes 13 bit times to transmit a single UART character (1 start bit, 8 data bits, 1 even parity bit, 1 stop bit and 2 idle bit times). Therefore, the transmission time of a single character is $1352 \mu sec$.

The timing of a standard message cycle is shown in Figure 6.3. First, the data frame is transmitted. The length of such a data frame is variable. Then the bus must be kept idle for 15 bit times (Idle 1;$1560 \mu sec$). After this idle time, the receiver of the data frame must respond with an acknowledge frame which is 1 octet long. After this acknowledge frame, the bus must be kept idle again (Idle 2). The idle time depends on the priority of the next frame. If the priority of the

message is `system` or `urgent`, the bus must be kept idle for at least 50 bit times ($5200\mu sec$). Otherwise, at least 53 bit times are mandatory ($5512\mu sec$).

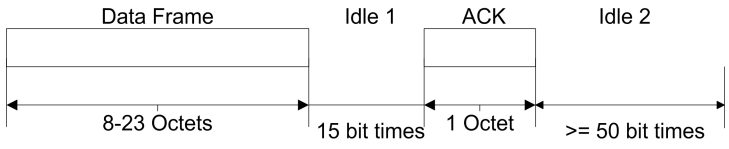


Figure 6.3: Message Cycle in EIB

As mentioned above, the length of a data frame is variable. Depending on the message type, a standard short frame is between 8 and 23 octets long. For a detailed description of the frame format see [39] or [60].

Based on this standard message cycle, the transmission time of the required messages can be calculated as follows:

- `A_Connect`:
  An `A_Connect` message is 8 octets long. This means that the transmission time of the data frame is $10816\mu sec$. Together with the acknowledge frame and the idle times 1 and 2, the transmission of an `A_Connect` messages takes $18952\mu sec \simeq 19ms$ (high priority).
- `A_Authorize_Request`:
  Since an additional application control field (1 octet) and 5 octets user data (key + one additional octet) is necessary, an `A_Authorize_Request` is 14 octets long. Therefore, this message requires together with the acknowledge frame and the two idle times $27040\mu sec \simeq 27ms$ (high priority).
- `A_Authorize_Response`:
  An `A_Authorize_Response` message contains the valid access level. Therefore, such a message is 10 octets long (1 octet application control field and 1 octet for the current valid access level). Therefore, an `A_Authorize_Response` message requires together with the acknowledge frame and the two idle times $21632\mu sec \simeq 22ms$ (high priority).
- `T_ACK_PDU`:
  This data frame is used as transport layer acknowledge. Since the frame has no application data, a `T_ACK_PDU` is 8 octets long. Therefore, it has the same length as an `A_Connect` message and therefore it takes $18952\mu sec \simeq 19ms$ (high priority).

To verify whether a key is valid or not, the above messages must be trans-

mitted over the network. First, the attacker must send an A_Connect message (needs $19ms$). As the same connection can be used during the whole attack, the connection must be established only once. Therefore, this message can be ignored for further calculations. To verify the key, an A_Authorize_Request message must be sent (needs $27ms$). This message is acknowledged using a T_ACK_PDU packet (19ms). Then, it sends an A_Authorize_Response which needs $22ms$. To avoid a retransmission, the attacker must also response with a T_ACK_PDU (19ms). This means that testing one key needs about $87ms$. Since a key is 32 bits long, 2^{32} different keys exist. Therefore, testing all possible keys will take $2^{32} * 87ms \simeq 12years$. Since the statistical expectation is 50%, there is an average of 6 years required.

It is important to note that the calculation mentioned above assumes that no collisions occur. This means that no other messages may appear which is very unrealistic in a working system. Additionally, the processing time of the requests is also ignored.

As shown in Section 6.1.4, the management tool ETS uses only one access key for the whole network. It is not possible to specify a separate key for each BCU. Therefore, it is possible to reduce the time it takes to find a valid key by performing a parallel attack. To achieve a parallel attack, several BCUs are attacked at the same time. It is important to note that the BCUs must be located in different network segments. Otherwise, a single attack will consume the whole network bandwidth.

A brute force attack produces a lot of network traffic. This increased network traffic influences the behaviour of the whole network and therefore such an attack attracts attention. So, a brute force attack is not suitable to find a valid key.

6.1.2 Protocol Vulnerabilities

As mentioned in the previous section, a brute force attack takes too much time. Therefore, other possibilities have to be found.

As mentioned in Section 5.3, the security architecture of EIB/KNX is very rudimentary. EIB/KNX does not support security mechanisms that provide data confidentiality, integrity, freshness and authentication. It is only possible to protect a BCU 2 against unauthorised access. One of the biggest problems of the access control mechanism is that the keys are transmitted in plaintext over the network. Both the A_Authorize_Request service and the A_Key_Write service contain the authentication key in plaintext. If an attacker has access to the network, the attacker can simply intercept such a message. To achieve this, a net-

work sniffer is necessary. For example, an attacker could switch a BCU into bus monitoring mode[2]. In this mode, the BCU forwards all received network messages (independent of the destination address) to the PEI interface. So, the attacker is able to analyse the traffic. If an `A_Authorize_Request` or an `A_Key_Write` is transmitted over the network, the attacker can simply extract the key out of the received messages.

A program example of an EIB/KNX key sniffer is shown in Appendix B.1. To access the bus medium, the Linux BCU 1 kernel driver is used. This driver provides the opportunity to communicate with a BCU 1 via the PEI16 serial interface (for further details about this interface see [59]). To intercept the network traffic, the attached BCU is switched into bus monitoring mode. In this mode all received network messages are directly forwarded to the PEI interface. After an `A_Authorize_Request` or an `A_Key_Write` message has been received, the corresponding key is written to `<stdout>`.

Another drawback of the unicast communication service of EIB/KNX is that parallel connections are not supported. If one node has established a connection to a particular device, all other connection requests to this device are ignored. This means that a BCU cannot handle more than one connection at the same time. It is important to note that an open connection only blocks connection-oriented messages. Group message can always be received even if a management connection is established. An attacker can use this restriction to perform a DoS attack. To achieve such an attack, the attacker simply establishes a connection to the target of the attack. As long as the attacker keeps up the connection, no other device is able to open a connection. It is important to note that the attacker does not need to know a valid access key to open a connection. If the remote node does not send an `A_Authorize_Request`, the access level with the lowest privileges is assumed (for example, in BCU 2, access level 3 is reserved for access level without authorisation). All other incoming connection requests are ignored, even if the initiator of the request knows the key of an access level that has more privileges.

Another security flaw of the access control mechanism is that it is possible to inject messages into an already established connection. After the access key of the `A_Authorize_Request` has been verified, the desired access level is stored as current access level. This current access level is valid during the whole session. For the rest of the session, only the source address of a message is verified. If the source address of a message is identical to the source address which was used in

[2]The system software of a BCU offers different operation modes. The bus monitoring mode is one of these modes. For further details see [39].

the A_Connect message, the device accepts the message. If the addresses are not identical, the message is ignored. There is no other mechanism available that verifies the identity of the data origin. As the source address can be spoofed very easily, it is not possible to verify the authenticity of further messages. For example, after a node has changed the current access level using the A_Authorize_Request service, an attacker can simply spoof the source address and inject other management messages.

The weakness mentioned above can also be used to perform another form of DoS attack. Since the source address is not protected, the attacker can close the connection at any time using the A_Disconnect service. To achieve this, the attacker spoofs the source address[3] of the A_Disconnect message and sends this faked disconnect request to one of the communication participants. As the receiving device believes that the other device wants to tear down the connection, it will close the connection. An "EIB virus" could take advantage of this weakness. Consider, for example, an "EIB virus" which infects several BCUs. In addition to deleting the user application, a malicious user application is uploaded. This application intercepts the network traffic. If a node A sends an A_Connect message to another node B, the virus detects this and immediately sends an A_Disonnect message to B (with the spoofed source address of A). B will tear down the connection and so A is not able to send any further connection oriented messages. This means that if the virus infects one single BCU in a line, it is not possible to establish a connection to any other device. As soon as a device sends an A_Connect message, the virus immediately sends an A_Disconnect message which closes the connection immediately.

Another drawback is that the access control mechanism is only available for unicast communication. This means that only management services like A_Memory_Read or A_Property_Read can be protected against unauthorised invocation. There is no protection against malicious modification of group messages. For example, an attacker can simply send an A_Groupvalue_Write message to modify the value of a group object. To guarantee the security of an EIB/KNX system, it is essential to protect process data communication too.

[3]To fake the source address of a message, the physical address must be changed. To achieve this, the new address must be written to memory 0x117 (if a BCU is used).

6.1.3 BCU 2 System Software

After the access protection protocol was examined, the implementation of this mechanism has to be verified. To achieve this, it is necessary to take a closer look at the internal structure of the BCU 2 system software. Since the source code of the system software is not public available, the memory of the BCU 2 has to be read out.

As mentioned in [61], the program code of the system software is located at memory 0x5000 - 0x7fff. Additionally, the memory area 0x04e0-0x04ff is also reserved for the system software. It is important to note that the management service `A_Memory_Read` cannot be used to read out these memory areas. The message system of the system software protects these memory areas and so, an access via `A_Memory_Read` is not possible. Therefore, another possibility has to be found.

To read out the necessary memory areas, a BCU 2 user application was written. This application called `Memread` reads out a specified memory area. The memory content is split into several pieces. These pieces are encapsulated into group messages which are transmitted over the network. To get the memory content, the group messages are simply intercepted using an EIB network sniffer.

The source code of this application can be found in appendix B.2. The application `Memread` was written using the BCU SDK. This development environment provides programming tools which are based on the GNU utilities. Using the BCU SDK, the code fragments can be written in C. Additionally, the BCU SDK provides several tools which can be used to upload the image of the application (for further details see [27]).

To analyse the received memory dump, the byte code must be disassembled. To convert the machine code into assembler instructions, the tool `objdump` from the BCU SDK was used. To get the correct alignment of the instructions, it is essential to find the correct entry point. According to the specification of the MCU (BCU 2 contains a Motorola MC68HC705BE12 MCU[4]), the reset vector which is located at memory 0x7fff is loaded after power up. This means that the reset vector contains the address of the first instruction which is executed after power up. Regarding BCU 2, this reset vector contains the value 0x781e. This means that the first instruction which is executed after power up is located at 0x781e.

In addition to this reset vector, a MC68HC705BE12 MCU has seven different interrupt vectors. These interrupt vectors are located at memory 0x7ff0-0x7fffd.

[4]Microcontroller Unit

Each interrupt vector contains the address of the first instruction which is executed after the corresponding interrupt occurs.

Beginning from these entry points, the machine code can be disassembled. After the machine instructions were disassembled, the obtained assembler instructions were analysed. It is obvious that without detailed knowledge about the internal structure of the system software, the analysis of the raw assembler instructions is a very complex task. Therefore, it was very difficult to get useful information about the system architecture. [61] gives a brief introduction to the system architecture of the BCU 2 system software. It is important to note that this help file only provides information about the user relevant part of the system software (for example information about API callbacks, user timers, ...). Furthermore, the used MC68HC705BE12 MCU was specially designed for EIB/KNX. Therefore, it is very difficult to find details about the internal structure of the microcontroller (only a data sheet is available; for further details see [68]). For example, the exact location of the input registers (timer register, counter registers, ...) was not known.

Anyhow, the following information has been determined during analysis:

- Memory location of access keys:
  As mentioned above, the system software of BCU 2 supports four different access levels. Access level 3 is a synonym for free access which is always available. Therefore, no key can be specified for level 3. For each of the remaining three levels (see Table 5.1), a 4 byte key can be specified. These three keys are located at:
  - Key for Access Level 0: 0x049c-0x049f
  - Key for Access Level 1: 0x04a0-0x04a3
  - Key for Access Level 2: 0x04a4-0x04a7

  It is important to note that the user application has full access to these memory locations. As the microcontroller directly executes the instructions of the user application, the system software cannot restrict the access to the internal memory. Therefore, the user application can modify or delete the access keys.
- Current Access Level:
  After the device receives an `A_Authorize_Request`, it verifies the retrieved key. If the key is valid, the system software stores the desired access level as current access level. As mentioned in the previous section, a BCU cannot handle more than one connection at the same time. Therefore, only one current access level exists. This current access level is located at memory 0x87. Again, the user application can modify or read the current access level directly. Additionally, the system software provides an API function

called U_GetAccess (located at memory 0x5096). This routine copies the current access level to the accumulator. After a reset, the system software initialise this current access level. This is done by a subroutine that is located at 0x6388. This subroutine iterates through all three access levels and checks whether a key is set to empty or not[5]. After this check, the number of the lowest access level which has an empty key is stored as current access level.

As mentioned in Section 5.3, the A_Key_Write service can be used to change the key of a specified access level. To delete a key, the key must be set to the value 0xffffffff using an A_Key_Write message. During the analysis, another possibility was found. At memory 0x10d, the RunError register is located. This register contains error flags which indicate runtime errors. These different flags are set by the system software and can be used for error analysis purposes (for further details see [61]). This register can also be used to clear all three access keys. If the management service A_Memory_Write is used to clear the most significant bit of the RunError register, all three access keys are set to 0xffffffff. It is important to note that only access level 2 is necessary to use this feature. If a user has gained access using the key of access level 2, it is still possible to delete the keys of access level 0 and 1. This means that the key of access level 2 that has lower privileges than access level 0 and 1 is sufficient to perform a deletion of all keys. The analysis shows that the management tool ETS 3 uses this "magical feature". Therefore, it can be assumed that this mechanism is not an implementation flaw. A description of this feature cannot be found in any documentation. Neither the EIB/KNX specification nor the BCU 2 help file mention this feature.

6.1.4 Implementation Flaws in ETS 3

To configure and maintain EIB/KNX systems, a Microsoft Windows-based management tool called ETS is available. After the access control mechanism and the system software of the BCU 2 were analysed, it is necessary to examine how the management tool ETS uses this access protection mechanism. Therefore, the current available version 3 of ETS was analysed in detail.

First, it is important to note that only one single key can be specified (this key can be specified in the project properties). Using the ETS[6], it is not possible to

[5] A value of 0xffffffff denotes the empty key.

[6] In the rest of this chapter, the term ETS is used as a synonym for ETS version 3.0b.

define a separate key for each BCU 2. It is obvious that this approach is not very secure. If an attacker is able to retrieve this single key, the attacker is able to gain access to any BCU. A more secure solution is to use a separate key for each node.

Additionally, the ETS uses the same key for all three different access levels. It is not possible to define a separate key for each level. This means that the ETS does not distinguish between different access levels. However, if another tool is used to set separate keys for each access level, it is possible that the ETS cannot handle these BCUs anymore. For example, if the key for access level 1 or 2 is empty and the key for access level 0 is set to a non-empty value, the ETS cannot connect to such a BCU. If the ETS wants to authenticate to a BCU 2, it tries the empty key first. If the BCU 2 responses with access level 3, the ETS sends another authorisation request. This request contains the key which the user has defined in the project properties. If the BCU 2 responses with access level 2 or 1, the ETS aborts the transmission with an ”Internal Error” message.

Additionally, another software bug was found. The ETS is not able to set keys. If a new user application is uploaded, the ETS performs several steps. First, the ETS opens a connection to the target device. After the connection has been established, the ETS sends an `A_Authorize_Request`. As mentioned above, the ETS tries the empty key first. If the BCU responds with access level 3, the user specific key is sent. If the access is granted, the ETS initiates several initialisation operations (for example, initialise the memory control blocks, unloading the object table, ...). Afterwards, the ETS clears all keys by clearing the `RunError` register (for further details see Section 6.1.3). Then, the ETS uploads the image of the user application. After the upload of the image has been finished, the ETS closes the connection. It is important to note that the keys are not set by the ETS 3 after a new image is uploaded. It is obvious that this software bug must be fixed. Otherwise, an EIB/KNX network that is configured and maintained using ETS cannot use the access protection mechanism.

As mentioned in this section, the ETS does not provide the necessary facilities to support the access protection mechanism. Therefore, it is necessary to improve the current ETS implementation. Otherwise, the access protection mechanism cannot be used in a reasonable way.

6.2 A possible Approach of a Secure EIB/KNX Architecture (EIBsec)

As mentioned in this section, EIB/KNX is not suitable for providing a secure environment. The supported access control mechanism does not fulfil the requirements of a security critical application. As shown in Table 5.2, EIB does not guarantee data confidentiality, integrity and freshness. The access control mechanism provides only a very rudimentary form of protection. It is not suitable for most applications. Therefore, a stronger authentication mechanism is often necessary.

6.2.1 Previous Work

To satisfy the needs mentioned above, different security mechanisms have to be developed. At the TU München, a secure communication protocol called **Secure EIB (SEIB)** was developed (for further details see [62]). This security extension protects group communication against malicious attacks. It guarantees data confidentiality, integrity and freshness as well as data authenticity. It is based on the SNEP protocol which was explained in Section 4.2.2. SEIB uses AES to encrypt the content of group messages and a CRC checksum which is 32 bit long. To protect group communication against replay attacks, a 128 bit counter is used. This counter is long enough to guarantee that each counter value is unique during the whole life cycle of the system (see [62]). The used encryption scheme is similar to the one which is used in SNEP (see Figure 4.10). The encrypted counter is XORed with the plaintext which should be securely transmitted over the network. The advantage of this scheme is that the counter value is not transmitted over the network. So, the counter value does not waste any octets of the application part of the message (for further details see Section 4.2.2).

The frame format of a SEIB message is shown in Figure 6.4. As shown in this figure, only the transport control field, the application control field and the application data are encrypted. The rest of the message (including control field, source and destination address, network control field and layer 2 checksum) is transmitted in plaintext. The most important benefit of this approach is that encrypted SEIB messages can also be routed by devices which do not support the SEIB protocol. As the header information which is necessary to route a packet is available in plaintext, a line coupler that does not support SEIB is still able to route SEIB messages.

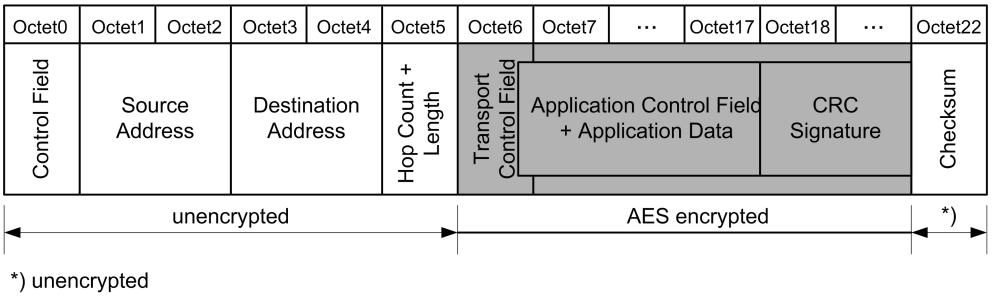


Figure 6.4: SEIB frame format

To use this secure group communication facility, each group needs a secret master key. This master key is necessary to generate the encryption key and the initial counter (for further details see Section 4.2.2). At installation time, this master key must be distributed to all group members. This distribution must be performed in a secure environment. Otherwise, an unauthorised user could obtain the master key during transmission. In SEIB, a master key can only be uploaded in physical mode. This means that the user must have a physical access to the particular device. For example, to prove the physical access to a device, the user must press the programming button of the device for several seconds. It is obvious that an unauthorised physical access to the device must be prohibited. Otherwise, an attacker could use this mechanism to upload a new master key.

As shown above, SEIB protects the process data communication against malicious attacks. There are still a few problems which remain unsolved. First, SEIB does not provide mechanisms to protect management communication. As mentioned above, the access control mechanism of EIB is not suitable to guarantee a secure transmission of management messages. Therefore, it is essential to protect management communication too.

Another drawback of SEIB is that the used key management is very rudimentary. SEIB does not support mechanisms to revoke insecure keys. Additionally, it is not possible to distribute a new key in an easy and secure manner. If the group key becomes public, the administrator must generate and distribute a new key manually. As mentioned above, the administrator must have physical access to upload a new key (physical mode). If the group has a lot of members, the distribution of a new key can be very time-consuming.

Another drawback is that the lifetime of the used keys is not limited. The different master keys are distributed at installation time. As long as the adminis-

trator does not distribute a new group key manually, a group key will remain valid. There is no possibility to limit the lifetime of the keys.

Due to these mentioned problems, the rest of this chapter will explain another possible solution called EIBsec. In addition to a secure process data communication, this solution protects management communication too. Additionally, it provides a key management infrastructure that provides the opportunity to generate and distribute new group and session keys in a secure manner.

6.2.2 System overview

As mentioned above, neither the access control protection of EIB/KNX nor security extensions like SEIB provide the required features to make EIB/KNX suitable for the use in security critical environments. Therefore, another possible secure architecture of EIB/KNX called EIBsec is explained in this chapter.

EIBsec should support the following mechanisms:

- data confidentiality, integrity and freshness
- authentication
- protection of management and process data communication
- key management
- basic intrusion detection
- basic update mechanism

To guarantee data confidentiality, integrity and freshness, EIBsec uses the encryption mechanism of SEIB. This encryption mechanism protects group communication against malicious attacks.

In contrast to SEIB, EIBsec protects management communication too. To provide a secure management communication, the transmission data is encrypted using session keys. During the session establishment, the secret key must be retrieved from a trusted third party (keyserver). It is important to note that the session key is only valid during a single session. If the session is closed, the key becomes invalid and so, it cannot be used anymore.

In addition to the encryption scheme mentioned above, EIBsec also provides an authentication mechanism. This mechanism can be used to verify the identity of both communication participants.

Another important feature of EIBsec is the integrated key management. The key management facility is responsible for:

- key generation and distribution:
  The used key management infrastructure must support mechanisms to generate and distribute the necessary keys in a secure manner. To achieve this,

EIBsec provides services which can be used to manage session keys (for secure management communication) as well as group keys (for secure process data communication).

- key revocation:
  If a secret key becomes insecure, it must be possible to revoke this key. Therefore, EIBsec supports a mechanism which can be used to revoke group keys[7].
- key lifetime limitation:
  Due to security reasons, the lifetime of a secret key should be limited. Therefore, EIBsec provides a mechanism to limit the lifetime of group keys[8].

In addition to the security mechanisms mentioned above, EIBsec supports a simple form of network-based intrusion detection as well as a basic update mechanism. These mechanisms can be used to observe the network traffic and to distribute software patches.

The most important component in EIBsec is called **Advanced Coupler Unit (ACU)** which is similar to a standard EIB/KNX line or backbone coupler. Compared to a standard coupler, an ACU performs additional tasks. An ACU consists of the following four building blocks:

- Coupler Unit
- Keyserver Unit
- Update Unit
- Intrusion Detection Unit

The **coupler unit** implements the standard coupler functionality. Like a line or backbone coupler, the coupler unit is responsible for routing the network traffic.

The **keyserver unit** implements the functionality of the necessary keyserver. The keyserver is responsible for the distribution and generation of session and group keys. It is also possible to revoke group keys and to limit their lifetime. Additionally, the group membership can be maintained (for further details see Section 6.2.4).

The **update unit** is responsible for distributing software updates. This feature can be used to correct implementation flaws of the system software of field devices (for further details see Section 6.2.6).

The **intrusion detection unit** performs the intrusion detection mentioned above.

[7] To revoke an insecure session key, the session is simply closed.

[8] Due to the fact that a session is only valid for finite time interval, the lifetime of a session key is also limited.

This unit is responsible for the detection of abnormal network traffic (for example DoS attacks) which may indicate a security attack. If such an abnormal situation has been observed, the intrusion detection unit informs other ACUs about this detected behaviour (for further details see Section 6.2.5).

It is obvious that such an ACU must be protected carefully against malicious attacks. To avoid a centralised approach, EIBsec uses a defence-in-depth solution. As mentioned in Section 5.3, the EIB/KNX network topology follows a three-level tree structure (lines, main lines, backbone). Therefore, it is practical to distribute the functionality mentioned above (keyserver, update server, intrusion detection unit) across this tree structured network. Figure 6.5 illustrates this approach. Each network segment (lines, main line or backbone) has its own ACU. In addition to the functionality of a normal line or backbone coupler, such an ACU distributes secret keys and software patches to its corresponding network segment. Additionally, it observes the traffic of the network segment to detect possible abnormal behaviour.

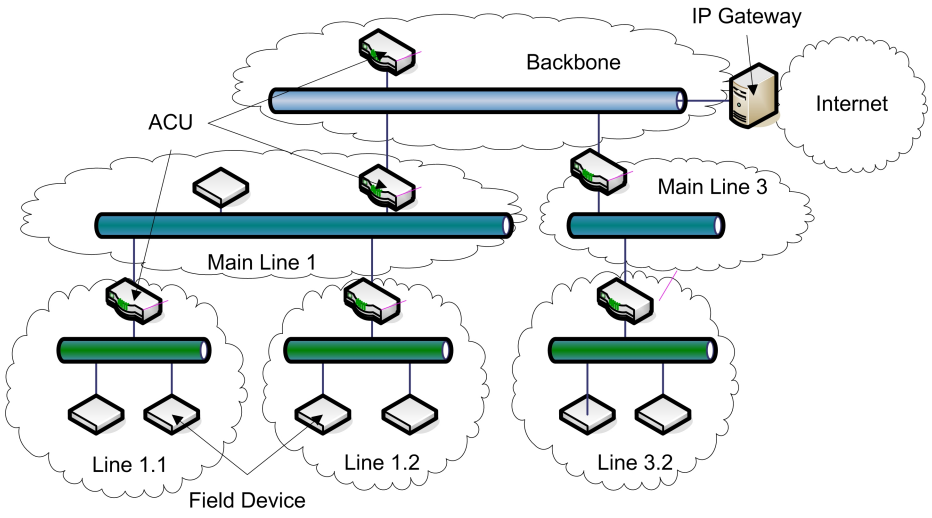


Figure 6.5: Topology of EIBsec

Figure 6.5 shows the topology of EIBsec. The rest of this chapter gives a detailed description of the different building blocks of EIBsec.

6.2.3 Secure Management Communication

To configure and maintain a field device remotely, it is necessary to establish a connection to the particular device. It is obvious that this unicast connection must be protect against security attacks. Additionally, it is necessary to verify the identity of both communication participants.

To satisfy the needs mentioned above, EIBsec uses an encryption mechanism which is similar to the one that is used in SEIB. In contrast to SEIB, EIBsec uses session keys to encrypt the messages. During session establishment, the necessary session key is retrieved from the keyserver. It is important to note that this session key is only valid during this single session. After the connection has been closed, the session key becomes invalid. To be sure that the communication participants are what they claim to be, the identities of both communication participants are also verified during this initial session establishment.

The protocol used is similar to the Needham-Schroeder protocol which is used in Kerberos. A detailed description of this protocol is given in [20]. Since the Needham-Schroeder protocol is vulnerable to malicious reuse of old session keys, a more secure variant is described in [20]. A variant of the latter is used in EIBsec.

Applying the protocol in [20] to EIB/KNX without further modification causes unacceptable overhead (see Figure 6.6). Suppose that an entity A wants to set up a connection to entity B. To initiate the session establishment, A sends an `A_Init_Connect_Request` message to B. B receives this request and sends back a nonce[9] N_1 (using an `A_Init_Connect_Response` message). This nonce is used to avoid a malicious reuse of an old session key. To protect this nonce against an unauthorised modification, B encrypts the response using its secret key (which is shared between B and S).

After having received this encrypted nonce, A sends an `A_SessionKey_Request` message to its keyserver S. Since S needs the encrypted nonce N_1, the session key request must contain this encrypted nonce. As A does not have the secret key of B, A is not able to decrypt the nonce. Therefore, it must transmit the encrypted `A_Init_Connect_Response` message as a whole to S. It is important to note that EIBsec uses AES-128 algorithm (for further details see Section 6.2.8) to encrypt messages. As AES-128 has a fixed block size of 128 bits (16 octets), each data block must be 16 octets long. Therefore, the encrypted nonce is also 16 octets long. Since the user data of the session key request is 22 octets long (16 octets encrypted nonce, 4 octets nonce of A and 2 octets address

[9] A nonce is 32 bits long. Therefore, 2^{32} different values exist. For further details see Section 6.2.8.

of B), A must send two messages to S.

After having received the message, the keyserver S generates a session key. This generated key must be distributed to both communication participants. It is important to note this session key is only sent to A. Instead of sending the session key to B, the keyserver sends a so called ticket together with the session key to the initiator A. This scheme guarantees that A does not send an encrypted message to B as long as B has not received the session key. If the keyserver S sends the session key to both participants, it is possible that A may want to set up a connection to B before B receives the session key. With the ticket scheme, this situation can be avoided.

This ticket looks as follows (N_2 denotes the nonce with was generated by B):

$$T = E(K_{BS}, A||K_{AB}||N_2))$$

Therefore, the response which the keyserver sends back to the initiator is (N_1 denotes the nonce with was generated by A):

$$E(K_{AS}, N_1||B||K_{AB}||T)$$

Such a ticket is 22 octets long (16 octets session key, 2 octets address of A and 4 octets nonce). As AES-128 uses a fixed block size of 128 bits, a ticket must be extended to 32 octets. As shown above, the response of the keyserver must contain the ticket and the session key. Since an EIB message can only contain 14 octets user data, this response must be split. To avoid replay attacks and to identify the response, each message must contain the nonce of A and the address of B. Therefore, there are only 8 octets left to transmit the session key and the ticket (14 octets user data - 4 octets nonce - 2 octets address). This means that the keyserver S must sent 6 messages to transmit the session key and the ticket to A.

After having received these messages, A can set up a secure connection to B. To achieve this, A sends an `A_Auth_Connect_Request`. This request contains a nonce and the ticket which A has received from the keyserver. As the ticket is 32 octets long and the nonce is 4 octets long, A must sent 4 messages.

After having received these messages, the identity of both communication participants must be proven. To achieve this, EIBsec uses the same mechanism and therefore it will be explained later in this section.

Figure 6.6 illustrates this protocol.

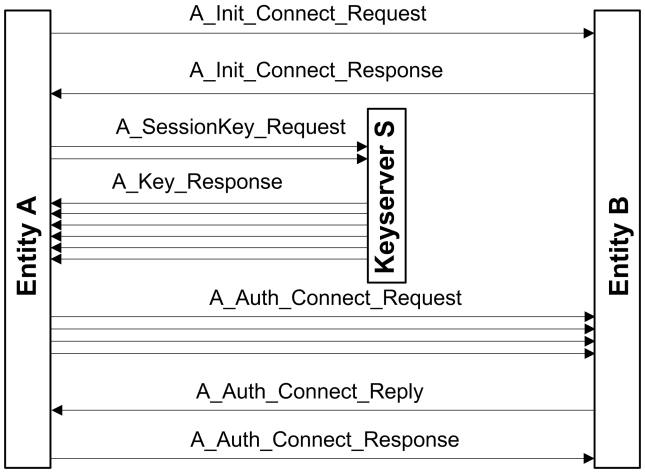


Figure 6.6: Protocol [20] in EIB

It is obvious that this scheme is too ineffective and therefore EIBsec uses another solution. Figure 6.7 illustrates the protocol which is used in EIBsec. Suppose, for example, an entity A wants to set up a secure connection to B. To achieve this, A and B need a session key. Therefore, A sends an `A_SessionKey_Request` (see Figure C.3) message to the corresponding ACU S. This message contains the individual address of B and a nonce N_1.

To avoid that a malicious user sends an old session key to B, a nonce is used. Compared to the protocol in [20], the ACU itself sends the `A_Init_Connect_Request` (see Figure C.1) to B. After having received this request, B sends a nonce N_2 to the ACU. Since B transmits N_2 directly to the ACU, A does not need to send the encrypted nonce to ACU. Therefore, only one message is required to send an `A_SessionKey_Request` primitive. After having received the `A_Init_Connect_Response` (see Figure C.2) message, the ACU generates a 128 bit session key K_{AB}.

This session key must be distributed to both communication participants. Instead of using a ticking scheme like in [20], the session key is sent to both communication participants. As the length of the user part of a standard EIB/KNX message (short frame format) is limited to 14 bytes, the session key must be split. Therefore, the lower bytes of K_{AB} are transmitted using the `A_Key_Response_Low` service (see Figure C.5) whereas the higher bytes are sent using an `A_Key_Response_High` message (see Figure C.6). To avoid replay attacks, these messages contain the corresponding nonce which was previously sent to the ACU.

Without such a nonce, an attacker could reuse an old session key.

Using this scheme, only 4 messages (2 for each participant) are required (see Table 6.1). Therefore, this approach reduces the amount of necessary messages. To avoid that A sends an `A_Auth_Connect_Request` message before B has received the session key, the ACU shall transmit the session key to B first. Additionally, the transmission to A shall be delayed. It is important to note that this delay can be set to a fixed value (in order of a few hundreds $msec.$). If A sends an encrypted message to B before B has received the session key, A can simply retransmit the message until the B has received the key.

After both entities have received the session key, A can establish a secure connection to B. To verify the identity of B, A sends an `A_Auth_Connect_Request` message (see Figure C.7) to B. This message contains a nonce N_3 and is encrypted using the session key K_{AB}. B receives this nonce and generates another nonce N_4. Afterwards, B performs a conversion of N_3 (for example, $N_3' = N_3 - 1$) and transmits the converted nonce N_3' together with the generated one N_4 back to A (using `A_Auth_Connect_Reply`; see Figure C.8). This conversion proves that B is capable to decrypt the message. After having received this encrypted message, A verifies if the converted nonce N_3' is correct. If N_3' is valid, the identity of B has been verified. To prove the identity of A, A converts the received nonce N_4 and transmits it back to B using a `A_Auth_Connect_Response` message (see Figure C.9). Again, A encrypts this message using the session key K_{AB}.

After both communication participants have proven their identity, A and B are able to communicate through a secure channel. To achieve this, all further messages are encrypted in counter mode. This counter mode is a special encryption mechanism which can be used after a session has been established. A detailed description is given in Section 6.2.8.

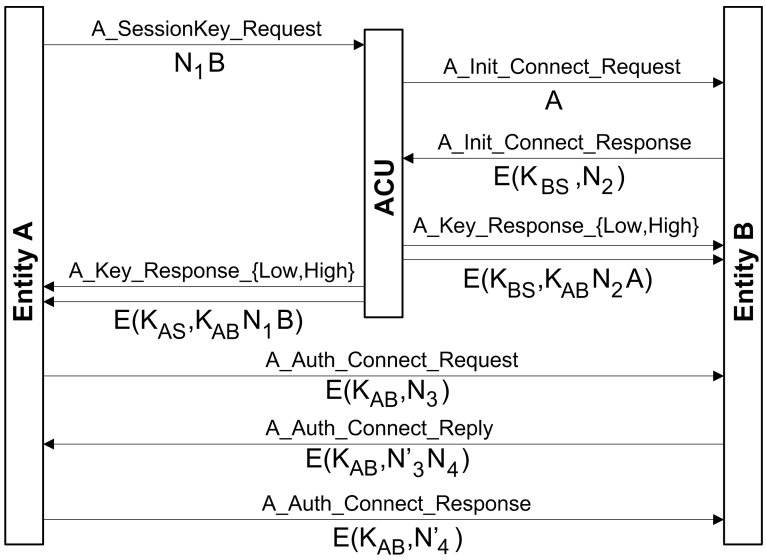


Figure 6.7: Authentication in EIBsec

Compared to the protocol in [20], the EIBsec authentication protocol is more suitable for the use in EIB/KNX. As shown in Table 6.1, the amount of necessary messages are reduced.

Message	**Protocol [20]**	**EIBsec**
A_Init_Connect_Request	1	1
A_Init_Connect_Response	1	1
A_SessionKey_Request	2	1
A_SessionKey_Response (to Initiator)	6	2
A_SessionKey_Response (to second P.)	0	2
A_Auth_Connect_Request	4	1
A_Auth_Connect_Reply	1	1
A_Auth_Connect_Rsponse	1	1
Summary	16	10

Table 6.1: [20] Protocol in EIB vs. EIBsec

It is important to note that this session key is only valid until the connection

is closed. A connection can be closed by sending an A_Disconnect message[10]. Additionally, a connection timeout (in order of a few seconds) should guarantee that the connection closes automatically.

As mentioned in Section 6.2.2, each network segment has its own keyserver. Therefore, each keyserver stores only the secret keys of the devices that are located in its segment. So, if the two communication participants which want to retrieve a session key are located in different segments, the ACU that handles the request does not have the private key of the second participant. To process the request, the ACU must get the key from another ACU. Therefore, each ACU must have the private key of its parent ACU and its child ACUs.

Figure 6.8 illustrates a possible solution how such a request can be forwarded. Suppose that entity A (with the individual address 1.1.1) wants to establish a session to entity B (with the individual address 1.2.2). To obtain a session key, A sends an A_SessionKey_Request to the corresponding ACU $S_{1.1.0}$ which has the individual address 1.1.0 (message 1). $S_{1.1.0}$ receives this request and sends an A_Init_Connect_Request to B (not included in Figure 6.8). B receives this request and generates a nonce. This nonce is encrypted using the secret key $K_{BS_{1.2.0}}$ ($S_{1.2.0}$ denotes the ACU which has the address 1.2.0). As B and $S_{1.1.0}$ are not located in the same line, $S_{1.1.0}$ does not have the secret $K_{BS_{1.2.0}}$. Therefore, $S_{1.1.0}$ is not able to decrypt the received A_Init_Connect_Response (not included in Figure 6.8).

To retrieve the secret key $K_{BS_{1.2.0}}$, $S_{1.1.0}$ sends an A_PrivateKey_Request (see Figure C.4) message to its parent ACU $S_{1.0.0}$ (message 2). This request contains a nonce and the address of the request secret key. To avoid an unauthorised modification, the request is encrypted using the secret key $K_{S_{1.1.0}S_{1.0.0}}$ which is shared by $S_{1.1.0}$ and $S_{1.0.0}$. $S_{1.0.0}$ receives the request and verifies whether it has the requested key. As B is not located in the main line 1, $S_{1.0.0}$ sends another A_PrivateKey_Request message to the child ACU $S_{1.2.0}$ (message 3). Since $S_{1.2.0}$ has stored the requested private key, it sends this private key back to $S_{1.0.0}$ using an A_Key_Response_Low (see Figure C.5) and an A_Key_Response_High (see Figure C.6) message (message 4). $S_{1.0.0}$ receives the secret key of B and transmits the key to $S_{1.1.0}$ (again using A_Key_Response_Low and A_Key_Response_High; message 5). After having received the secret key, $S_{1.1.0}$ is able to decrypt the A_Init_Connect_Response message (not included in Figure 6.8). Now, $S_{1.1.0}$ can generate a session key K_{AB} and transmit this key to both

[10]To avoid a malicious use of this A_Disconnect service, the message must also be encrypted (in counter mode).

communication participants (message 6 and 7). After A and B have received the session key, A can set up a secure channel to B (message 8). Figure 6.8 illustrates this example.

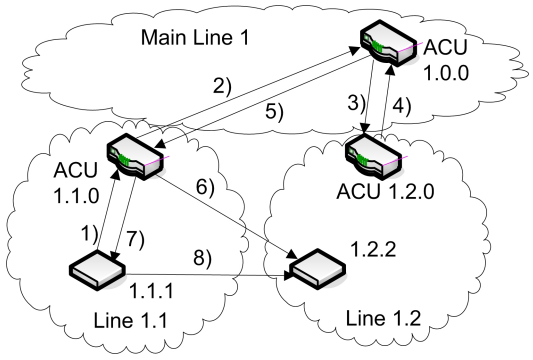


Figure 6.8: Forward Session Key Request

6.2.4 Secure Process Data Communication

To guarantee data confidentiality, integrity and freshness as well as data authentication of group messages, the group communication services must also be protected against malicious attacks. As mentioned at beginning of this section, SEIB provides such a protection. To ensure a secure transmission of group messages, the corresponding messages (`A_GroupValue_Read`, `A_GroupValue_Write` and `A_GroupValue_Response`) are encrypted in counter mode. To use this counter mode, a secret group key is needed. This group key is shared between the different group members. A detailed description of this counter mode is given in Section 6.2.8.

As explained at beginning of this section, the key management of SEIB is very rudimentary. Therefore, EIBsec provides additional services that help to manage the group membership.

Using EIBsec, it is not necessary to distribute the necessary group keys at installation time. Each field device can obtain the required group key from its corresponding ACU. Again, the different ACUs are responsible for managing the group key distribution.

To obtain a group key the following steps are necessary. Consider, for example, an entity A wants to send and receive group messages of a group G. To achieve this, A must retrieve the corresponding group key K_G. Therefore,

A sends an `A_Join_Group_Request` message (see Figure C.10) to its corresponding ACU. This message contains the group address and a nonce. The ACU receives this request and verifies whether the entity has the necessary access rights to join the group. With this optional mechanism, the ACU is able to manage the group membership. This means that it is possible to specify which entities are allowed to receive and transmit group messages. After the access rights have been verified, the ACU sends the group key together with the received nonce and the group address to A. Due to the fact that the group key is 128 bits long, two response messages (`A_Join_Group_Response_Low` see Figure C.11 and `A_Join_Group_Response_High` see Figure C.12) must be transmitted. Since A does not know the exact counter value necessary to encrypt a message in counter mode, the ACU also sends the current counter value to A. This is done by sending an `A_Group_Resync_Response_Low` (see Figure C.14) and an `A_Group_Resync_Response_High` (see Figure C.15) message. After having received the group key and the current counter value, A is able to receive and transmit group messages. It is important to note that the encryption and decryption of group messages is performed in counter mode (for further details see Section 6.2.8). Therefore, each member of a group must increment the counter after a message has been received or transmitted. Figure 6.9 illustrates this mechanism.

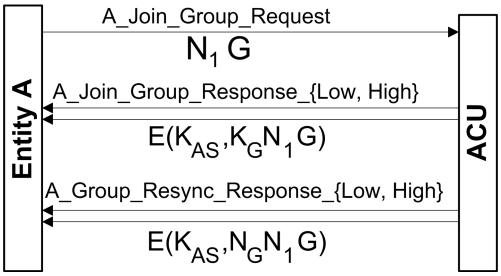


Figure 6.9: Group key retrieval in EIBsec

As mentioned above, encryption and decryption of group messages are performed in counter mode. It is essential that the group counters of all group members are synchronous[11]. If an entity loses a group message, the counter of the

[11]To achieve this, each group members must process all group messages of the corresponding group. This means that a write-only device (for example a light switch) must also receive group messages. If another group member transmits a group message, the write-only device must also increment its group counter.

entity will get loss of synchronisation. One possible solution is described in Section 6.2.8. If an entity recognises that its counter is not synchronous anymore (for example if the CRC signature is invalid), it can try a small number of counter increments. If this fails, it is possible to send an `A_Group_Resync_Request` (see Figure C.13) to its ACU. The ACU receives this request and sends back the current counter value using `A_Group_Resync_Response_Low` (see Figure C.14) and `A_Group_Resync_Response_High` (see Figure C.15). Again, a nonce is used to avoid replay attacks. To provide such a synchronisation mechanism, the ACU must keep track of the sent group messages. This means that the ACU must also count the received group messages.

EIBsec also provides the opportunity to revoke a group key. To achieve this, the ACU has to send a special group message. This `A_Group_Invalidate` (see Figure C.16) message informs all group members about the revocation of the group key[12]. After the ACU has sent such an `A_Group_Invalidate` message, the corresponding group key is not valid anymore. Therefore, each group member must request a new one. This is done by using the `A_Join_Group_Request` service mentioned above. It is important to note that it is the responsibility of each group member to get the new group key.

This mechanism can also be used to limit the lifetime of group keys. The ACU can periodically invalidate group keys using the `A_Group_Invalidate` service. Thus, it is possible to limit the lifetime of group keys and to force the generation of new ones.

6.2.5 Intrusion Detection Management

As mentioned at the beginning of this section, EIBsec also provides a basic intrusion detection service. To achieve this, each ACU has an integrated intrusion detection unit. This unit observes the network traffic and tries to discover abnormal behaviour[13].

Figure 6.10 illustrates how an ACU can inform its parent ACU about a detected intrusion. Consider, for example, ACU A has detected an abnormal behaviour. To inform its parent ACU B, A must establish a session with B. This is done by using the challenge-response mechanism of the authentication proto-

[12]To avoid the use asynchronous group counters and obsolete group keys, the ACU should also send an `A_Group_Invalidate` message to all group after a power off.

[13]The exact implementation of the detection mechanism is not specified in this version of EIBsec. This section shall only explain how an ACU can inform other ACUs about the detected intrusion.

col of EIBsec (for further details see Section 6.2.3 and Figure 6.7). Due to the fact that each ACU shares a secret key with its parent ACU, it is not necessary to exchange a session key. Therefore, all further messages are encrypted in counter mode using the shared secret key K_{AB}.

After the session has been established, A sends an `A_IDS_Alert` (see Figure C.20) message to B. In order to inform B about the detected intrusion. To achieve this, it contains an IDS code (2 octets long) as well as 8 octets additional information about the detected intrusion. After having received the `A_IDS_Alert` message, B could perform further countermeasures. For example, B could decouple the affected line until the threat has been eliminated by A.

To inform B that the problem has been solved, A can send an `A_IDS_Clear` (see Figure C.21) message to B. This message contains the IDS code which indicates eliminated intrusion. Figure 6.10 shows this concept.

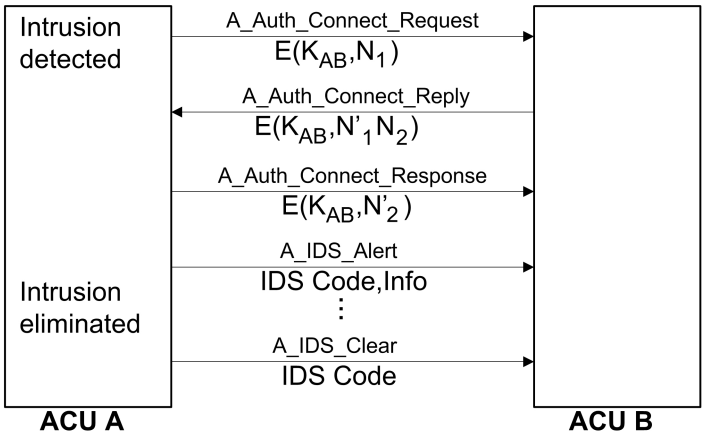


Figure 6.10: Intrusion Detection in EIBsec

6.2.6 Update Mechanism

If implementation flaws in the system software of a device have been found, it must be possible to correct these problems. As a building automation system can consist of hundreds or even thousands of devices, a manual software update is not easy to achieve. Therefore, a mechanism must be supported which provides the opportunity to distribute software patches in an easy and secure manner. EIBsec supports such an update mechanism. Using this mechanism, it is possible to distribute software patches automatically.

As mentioned at the beginning of this section, each ACU has an integrated update unit. This update unit receives software updates and distributes them to the corresponding devices. To initiate an update, the administrator must upload the software patches to the root ACU (see Figure 6.5). The root ACU can be the backbone coupler or an IP gateway. This root ACU distributes the software patch to its child ACUs. If a child ACU receives such a forwarded software update, it distributes the patches to the corresponding devices. Additionally, it forwards the update to its child ACUs.

Figure 6.11 illustrates how a software update can be transmitted to another device. It is important to note that the used mechanism to forward a software update to a child ACU and the mechanism to distribute an update to a field device are identical. To transmit a software update to another device B, the initiator A must establish a session to B. This is done by using the challenge-response mechanism of the authentication protocol of EIBsec (for further details see Section 6.2.3 and Figure 6.7). Due to the fact that each ACU shares a separate secret key with all child ACUs and devices of its network segment, the generation of a session key is not necessary. Again, all further messages are encrypted in counter mode.

After the session has been established, A sends an `A_Software_Update_Init` (see Figure C.22) message. This message contains the version number[14] of the system software which should be updated. The version number is used to identify the version of the system software. It avoids that a patch is applied to a wrong device. It is important to note that this version number and the mask version do not have to be identical. Additionally, the message contains the amount of software patches which will be sent in this session.

After this initialisation, the individual patches are transmitted to B. A software patch is sent using an `A_Software_Update_Patch` (see Figure C.23) message. This message contains the address and the length of the memory block as well as the new content of the memory which should replace the old one. As the length of an EIB/KNX message is limited, one single patch can only contain seven bytes data. If the receiver of the software update is an ACU, it distributes the patches to the devices. To verify whether the software version of a device is correct, the current valid software version of the target device (which is stored together with the secret key of the device) is compared with the version number of the `A_Software_Update_Init` message. If the version numbers are identically,

[14]This version number must uniquely identify the system software. To achieve this, several identification IDs are necessary. For example, this version number could consist of a manufacturer ID, a device ID and a software ID.

the ACU sends the software patch to the corresponding device using the same mechanism. Additionally, it forwards the update to its child ACUs.

If the receiver of the patches is a device which has the correct software version, the device replaces the specified memory content. After the software update has been finished, the device should be reset. This can be done by the device itself (for example by jumping to the reset routine) or by sending an `A_Restart` message. Figure 6.11 summarises such a software update.

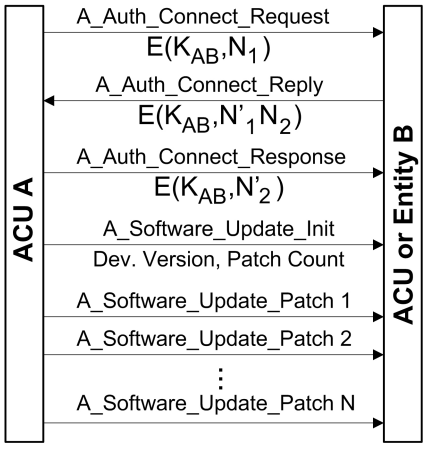


Figure 6.11: Software Update in EIBsec

It is obvious that a software update can only be performed if the system software is located in a programmable memory area. If the software software is stored in a one time programable ROM area, such an update is not possible. For example, the system software of the BCU 1 and 2 is located in such a ROM.

6.2.7 Initial Key Distribution

To retrieve a session key or a group key, it is necessary to communicate with the ACU in a secure manner. Therefore, the ACU encrypts the requested key using a symmetric algorithm. For example, the contents of `A_Key_Response_Low` and `A_Join_Group_Response_Low` are encrypted to avoid an unauthorised modification of the transmitted key. To perform such an encryption, each field node must share a secret key with the ACU.

At installation time, these shared secret keys must be distributed. This means that each node must obtain such a secret key. Otherwise, it is not possible to set

up a secure channel to the ACU. It is obvious that this initial key distribution must be performed in a secure environment. It must be avoided that an unauthorised user intercepts or modifies such a key during the initial key distribution.

It is also important to note that an attacker must not read out or change this secret key. This means that it must be avoided that an attacker uses the "key upload" mechanism to set a new secret key. Additionally, the read access to the memory where the key is stored must also be protected. For example, it must be guaranteed that an attacker is not able to use the `A_Memory_Read` service to read out the secret key.

As a secret key is 128 bits long and a standard EIB/KNX message can only contain 14 bytes of user data, the secret key must be split. To upload the secret key, EIBsec provides two services. `A_Set_SecretKey_Low` (see Figure C.17) can be used to upload the lower 8 bytes of the key whereas `A_Set_SecretKey_High` (see Figure C.18) is used to send the higher 8 bytes. In addition to the secret key, the remote user must send a password. This 6 byte password[15] is used to verify if the remote user is allowed to change the secret key. After having received an `A_Set_SecretKey_Low` or an `A_Set_SecretKey_High` message, the entity verifies whether the password is valid or not. If the password is correct, the new secret key is set. To change this password, the user must send an `A_Set_Password` (see Figure C.19) message. This message contains the old and the new password. If the old password is correct, the receiver of this request will change the password to the new value. To be able to change the password for the first time, the default password of a new BCU should be a standard value (for example 0xffffffffffff). It is essential that this default password is changed as soon as possible. Otherwise, an attacker could use this default password to change the secret key.

This explained mechanism has one drawback. The messages mentioned above are transmitted in plaintext. Thus, a malicious user could simply intercept the secret key or the password. Therefore, an additional protection mechanism is necessary. To upload a secret key in EIBsec, three different modes are available. Depending on the required security level, it is possible to choose separately one of these modes for each BCU. These three modes are:

- bus mode
- local mode
- direct mode

[15] As shown in Section 6.1.1, even 4 bytes are long enough to make a brute force attack infeasible.

If the **bus mode** is chosen, the EIB bus medium can be used to upload the secret key. Due to security reasons, the key distribution process must be performed in a secure environment. Otherwise, an attacker is able to intercept the key or the password during transmission. Therefore, this mode should only be chosen if it can be guaranteed that unauthorised users are not able to access the bus medium during key distribution. One possible solution is to connect the device directly to the management tool. In addition to the password protection mentioned above, the user must also have physical access to the particular device. One possible solution is to demand that the programming button of the BCU must be pressed for a few seconds. As mentioned at the beginning of this section, SEIB uses this scheme. In SEIB, a new secret key can only be uploaded, if the device is in physical mode.

If the **local mode** is chosen, the EIB bus medium cannot be used to upload a secret key. A device in local mode only accepts key upload requests via the local interface (for example via the PEI). It is obvious that this mode can only be chosen if the device has an accessible local interface. The upload process is identical to the one described above. Compared to the bus mode mentioned above, the secret key is never transmitted over the EIB bus and so, a user who has access to the network medium is not able to intercept the key during transmission. Therefore, if the bus medium cannot be protected against unauthorised users, this mode should be used instead of the bus mode.

The **direct mode** provides the strongest form of security. If a device is in direct mode, neither the EIB bus nor the local interface can be used to upload a secret key. This means that the manufacturer of the device must provide a opportunity to upload a new secret key. One possibility is to upload the secret key directly into the EEPROM of the microcontroller. However, it is obvious that this mode is very inflexible and hard to achieve.

6.2.8 Implementation

To protect transmission data against malicious attacks, the messages must be encrypted. Like SEIB, EIBsec only encrypts the transport control field, the application control field and the application data (see Figure 6.4). As mentioned above, this approach has the main advantage that couplers that do not support EIBsec are able to route such encrypted messages.

To encrypt the transmitted data, EIBsec provides two different modes. The first mode called **normal mode** performs only an encryption of the message. If this mode is used, the user itself is responsible for protecting the message against replay attacks (for example by adding a nonce). In normal mode, no counter

is included in the message. Additionally, the layer 7 CRC signature is also not added to the message. Therefore, all 14 octets user data can be used. Figure 6.12 illustrates this mode.

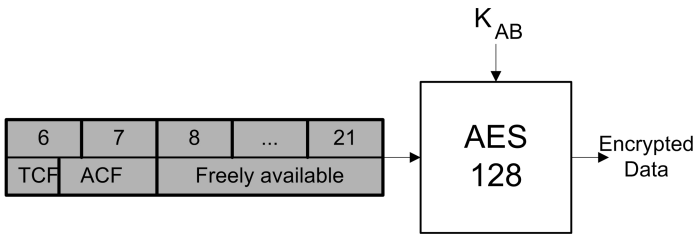


Figure 6.12: Normal encryption mode in EIBsec

The second encryption mode is called **counter mode**. This mode is similar to the counter mode which is used in SNEP and SEIB (see Figure 4.10). If this mode is used, a counter is automatically added. This counter protects the message against replay attacks. It is important to note that the counter must be long enough to guarantee that each message is unique during the whole life cycle of the system. Like in SEIB, the used counter is 128 bits long which is long enough to satisfy this need. To avoid the transmission of the counter value, the encryption scheme of SNEP is used. This means that the encrypted counter value is XORed with the message. Figure 6.13 illustrates this concept. Like in SEIB, a CRC 32 signature is also added to the user part of the message. Therefore, only 10 octets of user data can be used.

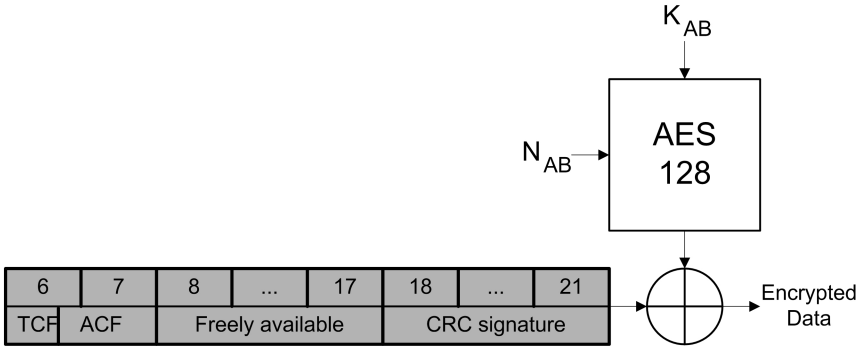


Figure 6.13: Counter encryption mode in EIBsec

To perform an encryption in the counter mode mentioned above, a secret key K and a initial counter N value are necessary. These are derived from a master key which is shared between the communication participants. Depending on the communication service, the session key or the group key is used as master key. The secret key and the initial counter value is calculated as follows (E denotes the encryption function of AES):

$$K = E(K_M, 1)$$
$$N = E(K_M, 2)$$

As mentioned in Section 4.2.2, both communication participants increment the counter after a message was encrypted or decrypted. It is important to note that the counter values of both participants must be synchronised. Otherwise, it is not possible to decrypt the received message correctly. However, sometimes messages may get lost and therefore the counter values will get asynchronous. To solve this problem, it must be possible to synchronous the counter. One possible solution is to try a small number of counter increments. For example, if an entity recognises that the current counter value N is not valid, it can try to decrypt the message using value $N + 1$ and $N + 2$. If this fails, another solution has to be found (for example, using the `A_Group_Resync_Request` as shown in Section 6.2.4).

In contrast to this 128 bit counter, a nonce is only 4 octets long. Such a nonce is used to avoid replay attacks if a message is encrypted in normal mode. For example, during session establishment several nonces are used to avoid replay attacks. In EIBsec, such a nonce is 4 octets long. This is long enough to guarantee that each nonce is unique during the whole lifetime of the system. Consider, for example, a device needs every second a new nonce. This means that the device requests a new session key every two seconds or a new group key every second. Since a nonce is 32 bits long, there are 2^{32} different values. This means that each device can run $136 years$ without using a nonce twice. Since a normal device will not request session and group keys in such an interval, a 32 bit nonce is long enough to guarantee that the nonce is unique during the whole lifetime of the system.

Regarding performance and memory requirements, field class devices have limited memory and limited computing power. Thus, asymmetric algorithms are too slow to run on such embedded devices. Therefore, it is essential to use a symmetric algorithm. As shown in appendix A, symmetric algorithms like triple-DES and AES are fast enough for even 8 bit microcontrollers. As mentioned in Section 3.2.1, DES is not secure anymore and therefore triple-DES or AES should

be used instead. Due to the fact that AES is faster and more secure, EIBsec uses AES. It is important to note that AES uses a fixed block size of 128 bits. Therefore, the length of the encrypted data should be a multiple of 128 (otherwise it must be extended).

A BCU 2 uses a Motorola MC68HC705BE12 MCU as microcontroller. As mentioned in [62], this MCU does not fulfil the performance and memory requirements. Therefore, another microcontroller has to be chosen. For example, in [62], a MSP430 is used instead.

6.2.9 Performance Considerations

Compared to normal group communication, the secure variant of EIBsec produces some overhead. This overhead has the following reasons:

- Each device must retrieve the necessary group key.
- To limit the lifetime of group keys, the ACU must invalidate the group key periodically.
- If the counter of a device gets asynchronous, the device must retrieve the current counter value.
- Every encrypted message must be 23 octets long.

If a device wants to receive and transmit encrypted group messages, the device must request the corresponding group key. To get such a group key, a communication with the ACU is necessary. As mentioned in Section 6.2.4, the device must send an `A_Join_Group_Request` message (15 octets long). After having received the request, the ACU sends back the desired group key using `A_Join_Group_Response_Low` (23 octets long) and `A_Join_Group_Response_High` (23 octets long). To be able to decrypt group messages in counter mode, the ACU must also send the current counter value to the device (`A_Group_Resync_Response_Low` (23 octets long) and `A_Group_Resync_Response_High` (23 octets long)). This means that a group key request causes 112 octets of network traffic (including 1 octet for each layer 2 acknowledge).

To determine the bus allocation time, the transmission time of these five mentioned messages must be calculated. These transmission times are calculated as follows (for further details about the timing of a standard message cycle in EIB/KNX see Section 6.1.1):

- `A_Join_Group_Request`:
  As mentioned above, an `A_Join_Group_Request` is 15 octets long. Together with the acknowledge frame (1 octet) and the idle times 1 ($1560\mu sec$)

and 2 ($5200\mu sec$), the transmission of this message takes $28392\mu sec \simeq 28ms$ (high priority).

- `A_Join_Group_Response_Low`, `A_Join_Group_Response_High`, `A_Group_Resync_Response_Low` and `A_Group_Resync_Response_High`: These messages are 23 octets long. Together with the acknowledge frame (1 octet) and the idle times 1 ($1560\mu sec$) and 2 ($5200\mu sec$), the transmission of such a message takes $39208\mu sec \simeq 39ms$ (high priority).

Due to the transmission times calculated above, the bus allocation time is $185ms$. It is important to note that the calculated time only includes the time it takes to transmit the necessary messages over the network. Other considerations like the execution time of the AES algorithm are not included in this calculation.

It is obvious that this key retrieval is only necessary, if the group communication is protected using EIBsec. If the unprotected group communication service of EIB/KNX is used, these messages are not necessary.

Due to security reasons, the lifetime of secret keys should be limited. Therefore, the ACU must invalidate the group key periodically. To achieve this, the ACU sends an `A_Group_Invalidate` message (23 octets). This message informs all group members about the invalidity of the current group key. After the ACU has invalidated the group key, it generates a new one. It is important to note that each group member is responsible for getting a new group key. This means that after a group key has been invalidated, each group member must request a new one. This is done by using the `A_Join_Group_Request` service mentioned above.

It is obvious that the network traffic which is caused by such a group key revocation depends on the amount of group members and the specified lifetime of the key. This means that each group key revocation causes $24 + 112N$ (N denotes the amount of group members) octets of network traffic (23 octets for `A_Group_Invalidate`, 1 octet layer 2 acknowledge and 112 octets for each key retrieval). Thus, if T_G denotes the lifetime of group key G in hours, a limitation of the lifetime of group key G causes $\frac{23+112N}{T_G} octets/h$ network traffic.

To calculate the bus allocation time, the transmission time of the required network messages must be calculated. As shown above, a message that is 23 octets long takes $39ms$. Since a single key request takes $185ms$, a group key revocation takes $39+185Nms$ (N denotes the amount of group members). Again, other considerations like execution time of algorithms are not included in this calculation.

In EIBsec, the group messages are encrypted in counter mode. To be able to decrypt a message which is encrypted in counter mode, the corresponding counter

must be synchronous. However, it is possible that the counter gets asynchronous (for example if a message gets lost). A possible solution is explained in Section 6.2.8. To compensate lost messages, the device tries a small number of counter increments. If this fails, the device can retrieve the current counter value by sending an `A_Group_Resync_Request` message to the ACU (15 octets). The ACU receives this request and sends back the current counter value of the requested group by using an `A_Group_Resync_Response_Low` (23 octets long) and an `A_Group_Resync_Response_High` (23 octets long) message. Thus, a counter resynchronisation request causes 64 octets network traffic (including one octet layer 2 acknowledge for each message).

The calculation of the bus allocation time is similar to the one described above. Since an `A_Group_Resync_Request` message is 15 octets long, it takes $28ms$ to transmit this message. To transmit the counter value, the ACU sends an `A_Group_Resync_Response_Low` and an `A_Group_Resync_Response_High` message. Each message is 23 octets long and therefore it takes $78ms$ to transmit the counter value ($39ms$ for each message). So, the transmission time of a counter resynchronisation request is $106ms$. Again, considerations like execution time of algorithms are not included.

Table 6.2 gives an overview about the results of the calculations mentioned above (N denotes the amount of group members).

Service	**Octets**	**Bus Allocation Time**
Key Request	112 (per node and group)	180 ms (per node and group)
Key Revocation	24+112N (per group)	39+180N ms (per group)
Counter Resynchronisation	64 (per node)	106 ms (per node)

Table 6.2: Network Traffic and Bus Allocation Time of EIBsec Services

As mentioned above, EIBsec uses AES-128. AES-128 has a fixed block size of 128 bits and therefore the content which should be encrypted must be at least 128 bits long. In EIBsec, only the transport field (octet 6), the application control field (octet 7) and application data are encrypted (see Figure 6.4). Since at least 16 octets are needed by AES-128 (due to the fixed block size), each message must contain 14 octets of user data. Even, if a group value of 1 bit is transmitted (for example, the state of a light switch), the group message must be 23 octets long

(instead of 8 octets).

Table 6.3 compares the network traffic and the bus allocation time of standard EIB/KNX group messages and EIBsec group messages (l denotes the message length in octets and t_{bus} denotes the bus allocation time of one message). In this table, the amounts of group messages which can be transmitted in one minute are also shown ($\frac{n}{min}$). Additionally, differences (column differences) between the possible standard EIB/KNX group messages per minute and the possible EIBsec messages per minute are also calculated.

Due to the fact that AES-128 has a fixed block size of 128 bits, an EIBsec group message is always 23 octets long. The length of a standard EIB/KNX message depends on the used data type. For example, if the data type `4-Octet Float Value` is used, a corresponding group message is 12 octets long.

Data Type	**standard EIB/KNX**			**EIBsec**			**Difference**
	l	t_{bus}	$\frac{n}{min}$	l	t_{bus}	$\frac{n}{min}$	
1 bit	8	19	3158	23	39	1538	-51%
8 bit	9	20	3000	23	39	1538	-49%
2 octets	10	22	2727	23	39	1538	-44%
4 octets	12	24	2500	23	39	1538	-38%

Table 6.3: Standard EIB/KNX group messages vs. EIBsec group messages

It is important to note that an EIBsec group message can only contain up to 10 octets. Due to the fact that group messages are encrypted in counter mode, a 32 bit CRC signature is included in each message (see Figure 6.13). Instead of 14 octets of user data, only 10 octets are available. This means that the data type `String` which needs 14 octets cannot be used in EIBsec. However, if the data type `String` is needed, the length must be restricted to 10 octets.

As shown in this chapter, EIBsec causes some additionally network traffic. The most critical service is the key revocation. The bus allocation time of the key revocation service is $38 + 185Nms$ per group. This means that if the building automation system has a lot of groups and the amount of group members is very high, this key revocation can produce a lot of network traffic. For example, a key revocation of a group with 5 group members needs about 1 second. To minimise an interference with the normal network traffic, the key revocation should be performed for each group separately. Additionally, the key revocation should be done when the system is not under heavy load (for example, late at night).

As shown in Table 6.3, each EIBsec group message is 23 octets long. Therefore, the amount of messages which can be transmitted in one minute is smaller. Anyhow, using EIBsec about 1500 group messages can be transmitted in one minute. This should be enough for most applications.

Chapter 7

Conclusion

At the beginning of this thesis, basic terms and definitions were explained. In Chapter 2, an overview about building automation systems was given. In the following Chapter 3, security concepts and the importance for building automation systems were explained.

After this brief introduction, different types of attacks were discussed. As mentioned in Chapter 4, a malicious user is able to attack the system from the outside or from the inside. Attacks from the outside (for example attacking an IP gateway) are well-known and several solutions exist. It is important to note that attacks from the inside are at least as dangerous as threats from foreign networks. Many systems rely on the assumption that isolation makes a system more secure. Therefore, today's building automation systems only provide rudimentary security mechanisms. But if an intruder has gained physical access to the control network, the intruder has full access to the system.

In Chapter 5, different building automation solutions were analysed. As mentioned in this chapter, LonWorks and EIB/KNX only provide rudimentary security features. The security architecture of LonWorks supports only a basic form of authentication. As shown in Section 5.1, this authentication mechanism is vulnerable to several security attacks. EIB/KNX uses an access control mechanism to protect devices against unauthorised access. The big problem of this mechanism is that the password is transmitted in plaintext. It is also important to note that neither LonWorks nor EIB/KNX can guarantee data confidentiality. Therefore, the security architecture of LonWorks and EIB/KNX is not suitable for most security critical application. In contrast to LonWorks and EIB/KNX, BACnet uses an encryption mechanism that guarantees data confidentiality. Additionally, BACnet supports an authentication mechanism which uses a central keyserver. Due to the

fact that the used algorithm (DES) is obsolete and that several security flaws were found in the authentication protocol, the scope of BACnet is also limited.

The last part of this thesis (Chapter 6) took a closer look at EIB/KNX. After a detailed analysis of the access control mechanism of EIB/KNX, a secure EIB/KNX architecture called EIBsec was described. This architecture provides the following features:

- secure management communication
- secure process data communication
- key management using distributed keyservers
- topology uses defence-in-depth approach
- optional intrusion detection and update mechanism

It is important to note that this solution has not been implemented yet. To evaluate the usability of this architecture, a detailed performance analysis should also be performed. Therefore, further development has to be done.

As mentioned in Section 6.2.5, the exact implementation of the intrusion detection mechanism is not defined in this version of EIBsec. In the current version, it is only specified how a device can inform other devices about a detected intrusion. Since many different intrusion detection mechanisms exist, the usability of these available solutions must be analysed. Therefore, further development has to be done in this field.

Since wireless technologies are getting more and more important, the security issues of these technologies must also be analysed. In EIB/KNX, a wireless solution called KNX Radio Frequency (RF) is available. It is important to note that such a wireless solution provides new opportunities for attackers. Therefore, it should be verified whether EIBsec provides the necessary security mechanisms to protect wireless communication against malicious attacks.

As mentioned in this thesis, security is getting more and more important in the building automation domain. Since most available solutions do not support the necessary mechanisms to satisfy today's security requirements, further development in this respect is necessary.

List of Figures

List of Tables

Bibliography

[1] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication Systems for Building Automation and Control," in *Proceedings of the IEEE*, vol. 93, no. 6, June 2005, pp. 1178–1203.

[2] C. Schwaiger and A. Treytl, "Smart Card Based Security for Fieldbus Systems," in *IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, September 2003, pp. 398–406.

[3] C. Schwaiger and T. Sauter, "A Secure Architecture for Fieldbus/Internet Gateways," in *IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, October 2001, pp. 279 – 285.

[4] P. Neumann, "Das Virtual Automation Network," *Computer&Automation*, vol. 2, pp. 66–70, 2005.

[5] ——, "Virtual Automation Network - Problems to be solved," in *IEEE Conference on Emerging Technologies and Factory Automation*, vol. 2, September 2003, pp. 3–6.

[6] ——, "Virtual Automation Network - Reality or Dream," in *IEEE International Conference on Industrial Technology*, vol. 2, December 2003, pp. 994– 999.

[7] W. Morr and W. Schmidt, "IT-Security und Leittechnik," *atp*, vol. 1, pp. 30–35, 2005.

[8] D. A. McKinnon, D. E. Bakken, and J. C. Shovic, "A Configurable Security Subsystem in a Middleware Framework for Embedded Systems," School of Electrical Engineering and Computer Science Washington State University, Tech. Rep., 2004.

[9] P. Bergstrom, K. Driscoll, and J. Kimball, "Making home automation communications secure," in *IEEE Computer*, vol. 34, no. 10, October 2001, pp. 50–56.

[10] P. Palensky and T. Sauter, "Security Considerations for FAN-Internet Connections," in *IEEE International Workshop on Factory Communication Systems*, September 2000, pp. 27–35.

[11] A. D. Wood and J. A. Stankovic, "Denial of Service in Sensor Networks," in *IEEE Computer*, vol. 35, no. 10, October 2002, pp. 54–62.

[12] M. Rahman, "Remote Access And Networked Appliance Control Using Biometrics Features," in *IEEE Transactions on Consumer Electronics*, vol. 49, no. 2, May 2003, pp. 348–353.

[13] *Building Automation and Control Systems (BACS) – Part 2: Hardware*, IS0 Std. 16 484-2, 2004.

[14] T. Zahariadis, K. Pramataris, and N. Zervos, "A Comparison of competing Broadband in Home Technologies," *Electronics & Communication Engineering Journal*, vol. 14, no. 4, pp. 133–142, August 2002.

[15] H. Ferreira, H. Grove, O. Hooijen, and A. H. Vinck, "Power Line Communications: An Overview," in *IEEE AFRICON 4th*, vol. 2, September 1996, pp. 558–563.

[16] J. Bray and C. F. Sturman, *Bluetooth 1.1: Connect Without Cables*, 2nd ed., M. Vincenti, Ed. Prentice Hall, 2002.

[17] (2005) IEEE 802.15.4. [Online]. Available: http://www.ieee802.org/15/pub/TG4.html

[18] (2005) Zigbee. [Online]. Available: http://www.zigbee.org

[19] W. H. Ware, C. P. Pfleeger, and S. L. Pfleeger, *Security in Computing*, 2nd ed. Prentice Hall, 1997.

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

[21] D. Dzung, M. Naedele, T. P. V. Hoff, and M. Crevation, "Security for industrial communication systems," in *Proceedings of the IEEE*, vol. 93, no. 6, June 2005, pp. 1152–1177.

[22] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 2002.

[23] M. Zimmermann and K. Dostert, "An Analysis of the Broadband Noise Scenario in Powerline Networks," in *4th International Symposium on Powerline Communications and its Applications*, April 2000, pp. 131–138.

[24] J.-P. Thomesse, "Fieldbus Technology in Industrial Automation," in *Proceedings of the IEEE*, vol. 93, no. 6, June 2005, pp. 1073–1101.

[25] (2005) GnuPG. [Online]. Available: http://www.gnupg.org/

[26] T. Wollinger, J. Pelzl, V. Wittelsberger, and C. Paar, "Elliptic and Hyperelliptic Curves on Embedded Microprocessors," in *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, August 2004, pp. 509–533.

[27] M. Kögler, "Free Development Environment for Bus Coupling Units of the European Installation Bus," Master's thesis, TU Wien, Institute of Automation, 2005.

[28] T. Elgamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in *IEEE Transactions on Information Theory*, vol. 31, no. 4, July 1985, pp. 469–472.

[29] R. L. Rivest and A. Shamir, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," in *Communication of the ACM*, vol. 21, no. 2, February 1978, pp. 120–126.

[30] J. Daemen and V. Rijmen, *The Design of Rijndael (AES- the Advanced Encryption Standard)*. Springer Verlag, 2002.

[31] *Data Encryption Standard (DES)*, FIPS Pub. 46-3, 1999.

[32] W. Diffie and M. E. Hellmann, "New Directions in Cryptography," in *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, November 1976.

[33] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, 1992. [Online]. Available: http://rfc.net/rfc1321.html

[34] *Secure Hash Standard (SHS)*, FIPS Pub. 180-2, 2002.

[35] J. F. Kurose and K. W. Ross, *Compuer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2001.

[36] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," in *Communications of the ACM*, vol. 21, no. 12, December 1978, pp. 993–999.

[37] G. Bruce and R. Dempsey, *Security in Distributed Computing*. Prentice Hall, 1997.

[38] E. A. Fisch and G. B. White, *Secure Computers and Networks: Analysis, Design and Implementation*. CRC Press LLC, 2000.

[39] *KNX Handbook 1.1 and KNX Standard Extensions*, Konnex Association, Brussels, 2004.

[40] NSA, "Defense in Depth: A Practical Strategy for Achieving Information Assurance in todays highly Networked Environments," NSA Security Recommendation Guides, Tech. Rep., 2003.

[41] M. R. Stytz, "Considering Defense in Depth for Software Applications," in *IEEE Security & Privacy Magazine*, vol. 1, no. 2, February 2004, pp. 72–75.

[42] T. Dierks and C. Allen, "Transport Layer Security," RFC 2246, 1999. [Online]. Available: http://rfc.net/rfc2246.html

[43] (2005) OpenVPN. [Online]. Available: http://openvpn.net/

[44] (2005) UPnP Forum. [Online]. Available: http://www.upnp.org/

[45] A. Treytl, T. Sauter, and C. Schwaiger, "Security Measures for Industrial Fieldbus Systems - State of the Art and Solutions for IP-based Approaches," in *IEEE International Workshop on Factory Communication Systems*, September 2004, pp. 201–209.

[46] N. Okabe, S. Sakane, K. Miyazawa, K. Kamada, A. Inoue, and M. Ishiyama, "Security Architecture for Control Networks using IPsec and KINK," in *Symposium on Applications and the Internet*, January 2005, pp. 414–420.

[47] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and rsa on 8-bit cpus," in *CHES 2004: 6th International Cryptographic Hardware and Embedded Systems*, August 2004, pp. 119–132.

[48] A. Perrig, R. Szewczyk, J. Tygar, Victorwen, and D. E. Culler, "SPINS: Security Protocols for Sensor Networks," in *7th Annual International Conference on Mobile Computing and Networking (MobiCom)*, July 2001, pp. 189–199.

[49] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Countermesure Against Connection Depletion Attacks," in *Network and Distributed System Security Symposium*, March 1999, pp. 151–165.

[50] B. Weiss, "Authentication Under Denial-of-Service Attacks," in *Communications and Computer Networks*, November 2002, pp. 134–139.

[51] *Neuron C: Programmer's Guide*, 7th ed., Echelon, 2003.

[52] *BACnet-A Data Communication Protocol for Building Automation and Control Networks*, ANSI/ASHRAE Std. 135, 2004.

[53] *Building Automation and Control Systems (BACS) - Part 5: Data Communication Protocol*, ISO Std. 16 484-5, 2003.

[54] *Control Network Protocol Specification*, ANSI/EIA/CEA Std. 709.1, 1999.

[55] *Tunneling Component Network Protocols Over Internet Protocol Channels*, ANSI/EIA/CEA Std. 852, 2002.

[56] D. G. Holmberg, "BACnet Wide Area Network Security Threat Assessment," NISTIR 7009, National Institute of Standards and Technology, Tech. Rep., 2003.

[57] J. Zachary, R. Brooks, and D. Thompson, "Secure Integration of Building Networks into the Global Internet," NIST GCR 02-837, National Institute of Standards and Technology, Tech. Rep., 2002.

[58] *KNX Specification*, Konnex Association Version 1.1, 2004.

[59] F. Praus, "An embedded and versatile KNX/EIB platform," Master's thesis, TU Wien, Institute of Automation, 2005.

[60] W. Kastner and G. Neugschwandtner, "EIB: European Installation Bus," in *The Industrial Communication Technology Handbook*, ser. The Industrial Information Technology Series. CRC Press, 2005, vol. 1, ch. 34.

[61] "BCU 2 Helpfile," Siemens, Tech. Rep., 2004.

[62] G. Westermeir, "Diversitäre Zugangs- und Sicherheitsmechanismen angewendet in automatisierten Gebäuden," Ph.D. dissertation, TU München, 2004.

[63] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing," HP Laboratories, Tech. Rep., July 2003.

[64] R. L. Rivest, *The RC4 Encryption Algorithm*, RSA Data Security, Inc., March 1992.

[65] A. Perrig, R. Canetti, J. Tygar, and S. Dawn, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," in *IEEE Symposium on Security and Privacy*, May 2000, pp. 56–73.

[66] (2005) Granzer Wolfgang EIB Development Website. [Online]. Available: http://eib.woif.org/

[67] W. Kastner and C. Troger, "Interfacing with the EIB/KNX: A RTLinux Device Driver for the TPUART," in *5th IFAC Conference on Fieldbus Systems and their Applications*, 2003, pp. 29–36.

[68] *Technical Data Sheet MC68HC705BE12 EIB/KNX*, Siemens, 2002.

[69] W. Kastner, G. Neugschwandtner, and M. Kögler, "Programable Fieldbus Nodes: A RAD approach to Customizable Applications," in *5th IEEE International Conference on Emerging Technologies and Factory Automation*, vol. 1, September 2005, pp. 1061–1064.

Appendix A

Performance of Cryptographic Algorithms

A.1 RSA

Data size: 200 Bits
Program: Part of GnuPG 1.0.7
CPU: H8/3048 19 Mhz 16-Bit
All values are taken from [46].

Key Length	**Encryption**	**Decryption**
1024	$\sim$ 0.65 sec.	$\sim$ 90 sec
2048	$\sim$ 1.5 sec.	$\sim$ 590 sec.

Table A.1: Performance of RSA - 16 Bit CPU

Data size: 200 Bits
CPU: ATmega128 8 Mhz 8-Bit
All values are taken from [47].

Key Length	**Encryption**	**Decryption**
1024	0.43 sec.	10.99 sec
2048	1.94 sec.	83.26 sec.

Table A.2: Performance of RSA - 8 Bit CPU

A.2 Diffie-Hellmann Key Exchange

Program: Part of GnuPG 1.0.7
All values are taken from [46].

Prime size	**public value**	**shared value**	**total**
1024	$\sim$ 190 sec.	$\sim$ 280 sec	$\sim$ 470 sec

Table A.3: Performance of Diffie-Hellmann Key Exchange

A.3 Symmetric Algorithms

Data size: 200 Bytes
CPU: DS80C390 36 Mhz 8-Bit
All values are taken from [46].

Algorithm	**Processing Time**
3DES	$\sim$ 500 msec.
AES	$<$ 100 msec.

Table A.4: Performance of 3DES and AES

A.4 Hash Functions

Data size: 200 Bytes
CPU: DS80C390 36 Mhz 8-Bit
All values are taken from [46].

Algorithm	**Processing Time**
MD5	$\sim$ 100 sec.
SHA1	$\sim$ 190sec.

Table A.5: Performance of MD5 and SHA1

Appendix B

Source Code of Algorithms

B.1 EIB Key Sniffer: sniff.c

```
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

#include <errno.h>
#include <fcntl.h>
#include <signal.h>

#include <unistd.h>

#define STANDARDDEVICE "/dev/mkoegler/eib0"

// file descriptor
int eib_fd=-1;

/*
 * Usage message
 */
void usage() {
  fprintf(stdout,"Usage: sniffkey [-d device]\n");
  exit(3);
}

/*
 * Closes the eib file descriptor
 */
void freeRessources() {
  if (eib_fd!=-1)
    {
      close(eib_fd);
      eib_fd=-1;
    }
}
```

```
/*
 * print error message
 */
void bailOut(char* message) {
  fprintf(stdout,"Error: %s \n",message);
  freeRessources();
  exit(5);
}

/*
 *captures SIGALRM & SIGINT
 */
void signalHandler(int sig) {
  switch (sig) {
    case SIGALRM:
        fprintf(stdout,"Error: no bcu found \n");
        freeRessources();
        exit(4);
        break;
    case SIGINT:
        freeRessources();
        exit(2);
        break;
    default:
        break;
  }
}

/*
 * main
 */
int main(int argc ,char **argv) {
  int c;
  unsigned char *buffer=malloc(32);
  int size;
  char *device;

  // get command parameters
  if (argc>3)
  {
      usage();
  }
  if (argc==1)
  {
      device=STANDARDDEVICE;
  }
  else
  {
    if ((c=getopt(argc, argv, "d:"))!=EOF)
    {
      switch (c) {
        case 'd':
            device=optarg;
            break;
        default:
            usage();
            break;
```

```
      }
    }
    else
    {
        usage();
    }
  }

  // open EIB device
  if ((eib_fd=open(device,O_RDWR))==-1)
  {
    bailOut("Can't open device");
  }

  // switch to busmonitor mode
  // To achieve this, use PC_SETVALUE (0x46) to write 0x90 to $60
  if (write(eib_fd,"\x46\x01\x0\x60\x90",5)!=5)
  {
    bailOut("Cant write to BCU\n");
  }

  while (1) {
    // read message
    if ((size=read(eib_fd,buffer,32))==-1)
    {
        bailOut("Cant receive packet from eib");
    }

    if ((size==18)&&((buffer[10]&3)==3)&&(buffer[11]==0xD1))
    {
        // A_Authorize_Request found
        printf("Found Request to Destination %d.%d.%d with key 0x%x%x%x%x\n",
            buffer[7]>>4,buffer[7]&0xF,buffer[8],buffer[13],
            buffer[14],buffer[15],buffer[16]);
    }
  }

  return 0;
}
```

B.2 BCU 2 application: Memread

Memread.config

```
Device {
  PEIType 0;
  BCU bcu20; // use bcu20 for a BCU 2.0
  Title "Memread";
  on_init init;

  FunctionalBlock {
    Title "Memread";
    ProfileID 10000;
    Interface {
```

```
      Reference { send };
      Abbreviation send;
      DPType DPT_String_8859_1;
    };

  };

  GroupObject {
    Name send;
    Type MAXDATA;
    Sending true;
    Title "Output";
    StateBased true;
  };

  Timer {
    Name timeout;
    Type UserTimer;
    Resolution RES_133ms;
    on_expire timer_interrupt;
  };

  include { "memread.c" };
};
```

Memread.c

```
int i;
static unsigned char *ptr;

void init() {
  ptr=0x5000;
  timeout_set(5);
}

void timer_interrupt() {
  if (ptr>0x7FFF)
    return ;

  send[0]=0;
  send[1]=((int)ptr>>8)&0xFF;
  send[2]=(int)ptr&0xFF;

  for (i=0;i<10;i++,ptr++)
    {
      send[i+3]=*ptr;
    }
  send_transmit();

  timeout_set(5);
}
```

Memread.config.ci

```
<?xml version="1.0"?>
<DeviceConfig
xmlns:ns1="http://www.w3.org/2001/XMLSchema-instance"
xmlns=
"http://www.auto.tuwien.ac.at/~mkoegler/eib/xml/configdesc.xsd"
version="0.0.0"
ns1:schemaLocation=
"http://www.auto.tuwien.ac.at/~mkoegler/eib/xml/configdesc.xsd">
  <ProgramID>XXX</ProgramID>
  <PhysicalAddress>1.1.98</PhysicalAddress>
  <InstallKey>FFFFFFFF</InstallKey>
  <Key id="0">FFFFFFFF</Key>
  <Key id="1">FFFFFFFF</Key>
  <Key id="2">FFFFFFFF</Key>
  <GroupObject id="id1">
    <!--Output-->
    <Priority>low</Priority>
    <SendAddress>0/0/1</SendAddress>
  </GroupObject>
</DeviceConfig>
```

Appendix C

EIBsec

C.1 Additional EIBsec Commands

Octet 6								Octet 7								
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
Secure Management Communication																
						1	1	0	0	0	0	0	0	0	1	A_Init_Connect_Request
						1	1	0	0	0	0	0	0	1	0	A_Init_Connect_Response
						1	1	0	0	0	0	0	0	1	1	A_Auth_Connect_Ticket
						1	1	0	0	0	0	0	1	0	0	A_Auth_Connect_Request
						1	1	0	0	0	0	0	1	0	1	A_Auth_Connect_Response
						1	1	0	0	0	0	0	1	1	0	A_SessionKey_Request
						1	1	0	0	0	0	0	1	1	1	A_PrivateKey_Request
						1	1	0	0	0	0	1	0	0	0	A_Key_Response_Low
						1	1	0	0	0	0	1	0	0	1	A_Key_Response_High

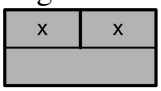
Table C.1: SEKA Commands Part 1

Octet 6								Octet 7								
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
Secure Process Data Communication																
	1	1	0	1	0	0	0	0	0	1	A_Join_Group_Request					
	1	1	0	1	0	0	0	0	1	0	A_Join_Group_Response_Low					
	1	1	0	1	0	0	0	0	1	1	A_Join_Group_Response_High					
	1	1	0	1	0	0	0	1	0	0	A_Resync_Group_Request					
	1	1	0	1	0	0	0	1	0	1	A_Resync_Group_Response_Low					
	1	1	0	1	0	0	0	1	1	0	A_Resync_Group_Response_High					
	1	1	0	1	0	0	0	1	1	1	A_Invalidate_Group					
Initial Key Distribution																
	1	1	1	0	0	0	0	0	0	1	A_Set_SecretKey_Low					
	1	1	1	0	0	0	0	0	1	0	A_Set_SecretKey_High					
Intrusion Detection																
	1	1	1	0	0	0	0	1	0	1	A_IDS_Alert					
	1	1	1	0	0	0	0	1	1	0	A_IDS_Clear					
Software Update																
	1	1	1	0	0	0	1	0	0	1	A_Software_Update_Init					
	1	1	1	0	0	0	1	0	1	0	A_Software_Update_Patch					

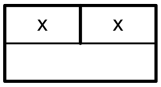
Table C.2: SEKA Commands Part 2

C.2 Frame Formats

Legend:



Octets are encrytped



Octets are NOT encrytped

C.2.1 Secure Management Communication

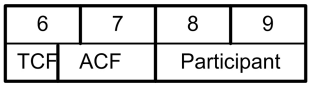


Figure C.1: A_Init_Connect_Request

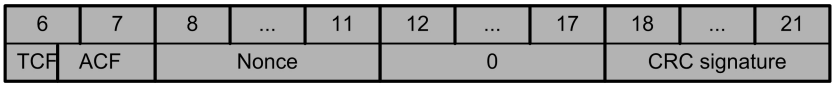


Figure C.2: A_Init_Connect_Response

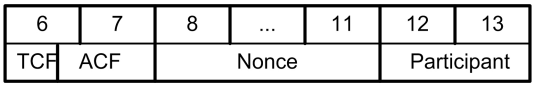


Figure C.3: A_SessionKey_Request

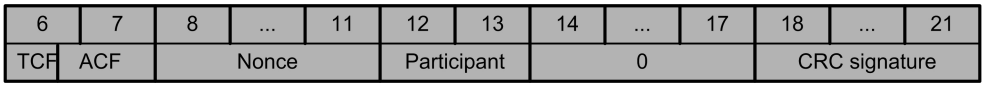


Figure C.4: A_PrivateKey_Request

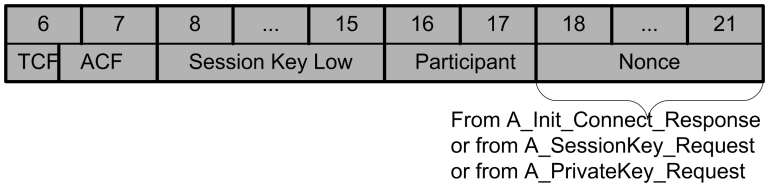


Figure C.5: A_Key_Response_Low

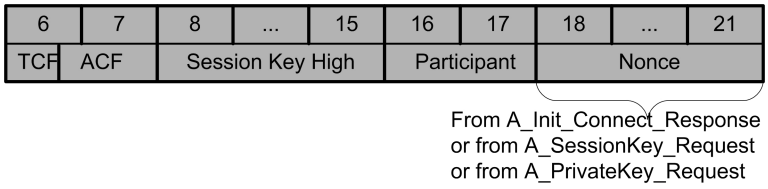


Figure C.6: A_Key_Response_High

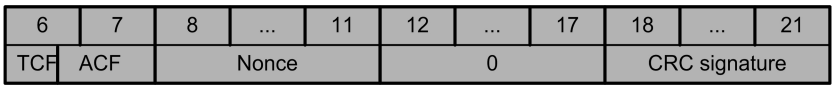


Figure C.7: A_Auth_Connect_Request

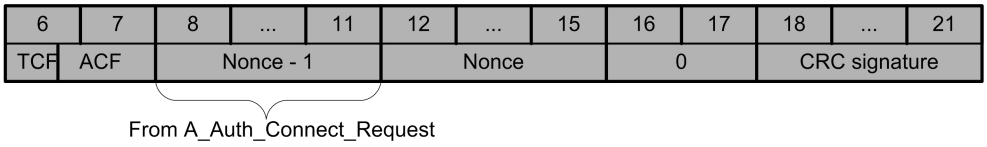


Figure C.8: A_Auth_Connect_Reply

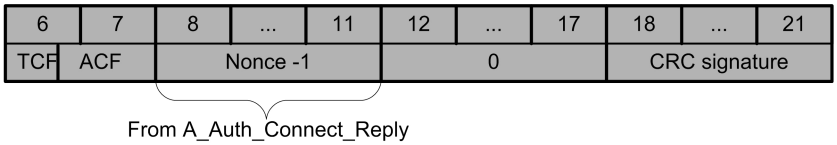


Figure C.9: A_Auth_Connect_Response

C.2.2 Secure Process Data Communication

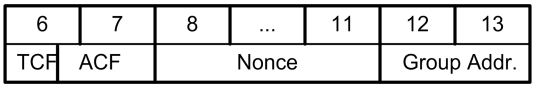


Figure C.10: A_Join_Group_Request

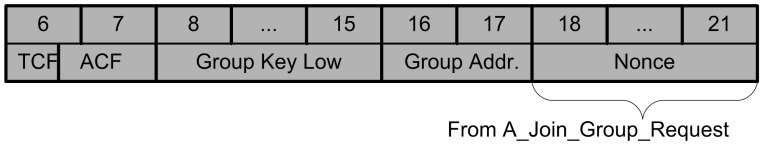


Figure C.11: A_Join_Group_Response_Low

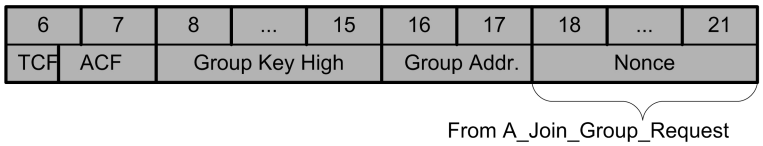


Figure C.12: A_Join_Group_Response_High

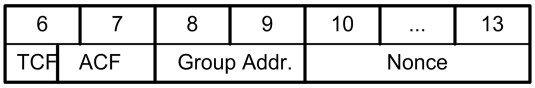


Figure C.13: A_Group_Resync_Request

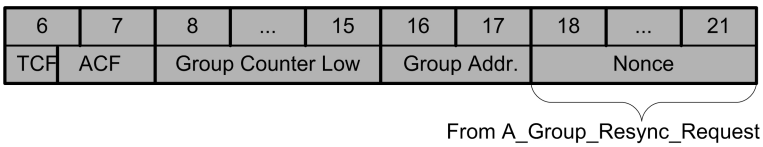


Figure C.14: A_Group_Resync_Response_Low

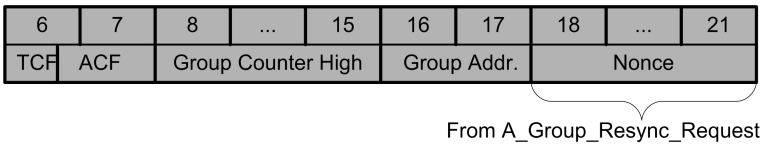


Figure C.15: A_Group_Resync_Response_High

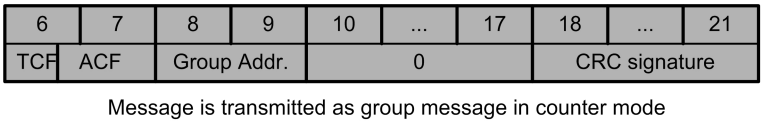


Figure C.16: A_Group_Invalidate

C.2.3 Initial Key Distribution

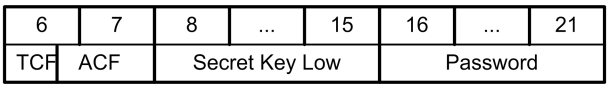


Figure C.17: A_Set_SecretKey_Low

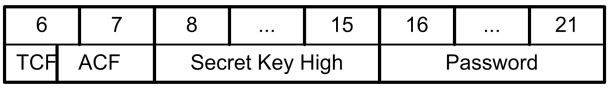


Figure C.18: A_Set_SecretKey_High

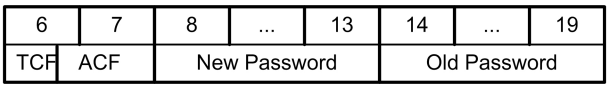


Figure C.19: A_Set_Password

C.2.4 Intrusion Detection

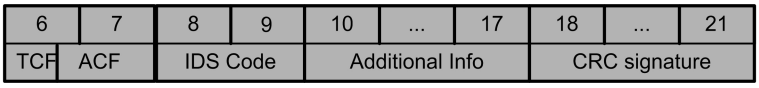


Figure C.20: A_IDS_Alert

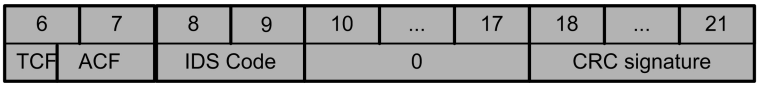


Figure C.21: A_IDS_Clear

C.2.5 Software Update

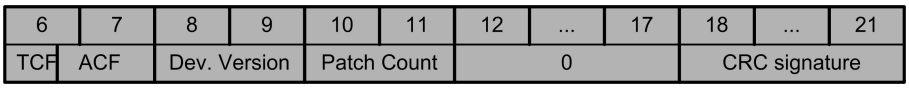


Figure C.22: A_Software_Update_Init

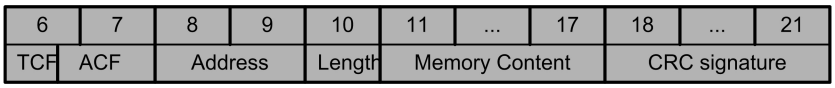


Figure C.23: A_Software_Update_Patch