Ref: 2018/II3/.. Academic Year: 2017-2018

Ministry of Higher Education and Scientific Research
Manouba University
National School of Computer Science

Graduation Project Report

Presented in the purpose of obtaining

Engineering Degree in Computer Science

By

Okba ZOUEGHI

Subject

End-to-end DTLS Based Security Layer for SafetyNETp

Host Company: Institute of Reliable Embedded Systems and Communication Electronics
Responsible: Prof. Dr.-Ing. Axel Sikora
Pedagogic Advisor: Dr. Chadlia Jerad
Supervisor: Dipl.-Phys. Andreas Walz
Tel/Fax: +49 0781 205-416
Website: http://ivesk.hs-offenburg.de

Signatures

Prof. Dr. Ing. Axel Sikora (Scientific Director)

Dr. Chadlia Jerad

Acknowledgments

I have the pleasure to express my sincere gratitude to all the people who helped me and who made the successful completion of this project possible.

I would like to offer my special thanks to Prof. Dr.-Ing. Axel Sikora for his trust and confidence and for giving me the opportunity to join the Institut für verlässliche Embedded Systems und Kommunikationselektronik (ivESK).

I am highly indebted to my supervisor Dipl.-Phys. Andreas Walz for his guidance, encouragement and his highly valuable and constructive suggestions during the realization of this work. I extremely appreciate the interesting discussions we have had and I am highly thankful for his willingness to give his time so generously.

I would like to express my deep gratitude for my pedagogic supervisor Dr.-Ing. Chadlia Jerad for her supervision, assistance, support and help as well as her important remarks and advice that helped to complete this project.

I would like also to thank the jury members for their time and efforts while reviewing my work. Besides, I am thankful to every Professor who helped me in my academic career.

This work would not be possible without the support of my family, I would like to especially thank my mother Faouzia, my sister Baraa and my brother Souhaib for their help and encouragement.

Last but not least, my deepest gratitude goes to Boutheina for her encouragement and support during this project.

Abstract

In today's automotive and industrial networks, data transmission is, for the most part, performed without any special security measures. This security policy weakness can create unforeseen security threats to the network, the network resources, and the data. SafetyNETp is an Industrial Ethernet protocol used in system automation technology which does not provide any security measures for protecting the network. The lack of security measures makes SafeyNETp vulnerable to several attacks. The work presented in this project consists of designing and implementing an end-to-end DTLS based security layer for SafetyNETp.

Résumé

Aujourd'hui, au sein des réseaux automobiles et industriels, la transmission de données est le plus souvent réalisée sans mesures de sécurité spéciales. Cette faiblesse peut créer une menace de sécurité imprévue pour le réseau, les ressources du réseau et les données. SafeyNETp est un protocole basé sur l'Ethernet industriel utilisé dans l'industrie qui ne fournit aucune mesure de securité ce qui le rend vulnérable à plusieurs attaques. Le travail présenté dans ce projet consiste à concevoir et implémenter une couche de sécurité basé sur le protocole DTLS qui permet d'assurer une sécurité de bout en bout.

مُلَخَص

في شبكات الإتصالات المستعملة في السيارات والصناعات اليوم، يتم نقل البيانات في معظم الحالات دون أي تدابير أمنية خاصة. ضعف هذه السياسة الأمنية يمكن أن يخلق تهديدات على أمن الشبكة و الموارد و البيانات. العمل المقدم في هذا المشروع يكمن في جزئين، في الجزء الاول نقوم بدمج بروتوكول حماية طبقة النقل DTLS عبر شبكة SafetyNETp ، في الجزء الثاني ، نقوم بتقييم ذلك الدمج واختبار البرنامج في بيئة واقعية. من أجل تحقيق هذين الجزئين ، نحن بحاجة لدراسة SafetyNETp وتطبيقها ، ودراسة DTLS بروتوكول ومن ثم العثور على التطبيقات الحالية المتوفرة ل DTLS و من خلال ذلك نقوم بإختياري التطبيق الأنسب لنظامنا و نقوم أخيراً بالدمج.

List of Abbreviations

AES	Advanced Encryption Standard
AH	Authentication Header
CA	Certificate Authority
CAN	Controller Area Network
CSMA	Carrier Sense Multiple Access
DES	Data Encryption Standard
DH	Diffie-Hellman key exchange
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
DAAD	Deutscher Akademischer Austauschdienst
ESP	Encapsulating Security Protocol
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IoT	Internet of Things
IKE	Internet Key Exchange
IPsec	Internet Protocol Security
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IMAP	Internet Message Access Protocol
ivESK	Institute of reliable Embedded Systems and Communications Electronics
LIN	Local Interconnect Network

MAC	Message Authentication Code
MS	Master Secret
OSI	Open Systems Interconnection
PKI	Public Key Infrastructure
PMS	Pre-master Secret
POP	Post Office Protocol
RTFN	Real Time Frame Network
RTFL	Real Time Frame Line
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TTL	Time To Live
TLS	Transport Layer Security
UDP	User Datagram Protocol
VPN	Virtual Private Network

Table of Contents

List of Figures

List of Tables

General Introduction

Nowadays, the industry is going more and more towards automation technology which is about automating industrial plants and their machines to such an extent that they independently do a job without the involvement of people. Although humans can control machines and industrial plants, they would not intervene in their work process. Thus, when a work process starts, human errors are eliminated. As a result, there is a significant increase in the quality of the products created.

The modern automated systems and field instruments currently being used are no longer mere mechanical devices, they are mostly controlled and managed by digital computers[1]. The majority of those systems and devices have not been designed with security measures[2]. In fact, many years ago, the motivation and the interest for hackers to compromise them was little or even inexistent. However, nowadays with new concepts such as Industrial Ethernet, Connected-Cars, Autonomous Vehicles and Internet of Things (IoT), the motivation for attacking those systems is increasing significantly.

Recently, multiple studies [1] demonstrated that the risk exists for automated systems to be infiltrated by malicious attackers that are potentially able to gain access via several methods. Despite this concrete threat, most of the industrial communication protocols do not have any built-in provisions to prevent or mitigate these attacks and no security aspect is part of the software or hardware architecture development process.

Current industrial communication protocols, including SafetyNETp, SafetyBUSp, Controller Area Network (CAN), and Local Interconnect Network (LIN), are vulnerable to attacks. Actually, they enable unauthorized access in a relatively straightforward manner given that all the communications are performed with no authentication [2]. However, we are convinced that security can be taken into account in the early phases of the development cycle of automated systems, both by implementing security protocols which can assure the authentication of the sender, the integrity of the message and the availability of the network, as well as by enforcing software programming standards that prevent software defects.

Transport Layer Security (TLS) is the most widely deployed protocol for securing network traffic. It is widely used for protecting Web traffic, for instance, HyperText Transfer Protocol Secure (HTTPS) the secure version of HyperText Transfer Protocol (HTTP) is secured using TLS. TLS is also used for securing e-mail protocols such as Internet Message Access Protocol (IMAP) and Post Office

Protocol (POP). The primary advantage of TLS is that it provides in a transparent way confidentiality, integrity, and authenticity to a given connection-oriented channel. Thus, it is easy to secure an application protocol by inserting TLS between the application layer and the transport layer [3]. However, TLS must run over a reliable transport channel typically the Transmission Control Protocol (TCP). Therefore, it cannot be used to secure unreliable datagram traffic. The requirement for datagram semantics automatically prohibits the use of TLS and for this purpose, there exists Datagram Transport Layer Security (DTLS) which provides TLS's services for unreliable datagram protocols such as the User Datagram Protocol (UDP) [4].

Alongside TLS, Internet Protocol Security (IPsec) is also a commonly used protocol for securing network traffic. It is implemented in the network layer and it provides security for both IPv4 and IPv6 traffics. Operating on the network layer makes IPsec transparent and independent from the application layer, this, in fact, eases the security management in multiapplication environments. IPsec is mostly used for creating a Virtual Private Network (VPN).

In this context, part of the funding program from Deutscher Akademischer Austauschdienst (DAAD), we have integrated the Institute of reliable Embedded Systems and Communications Electronics (ivESK) to perform our end of studies project. Our project consists of designing and developing a security layer for SafetyNETp which is a protocol for Ethernet-based Fieldbus communication in automation technology. Moreover, after the development, a test of the implementation is required to get an idea about the potential behavior of secured SafetyNETp networks in a realistic environment.

During the first chapter, we see a brief presentation of the host organism then we present the problem description as well as the objective and the proposed solution. Throughout the second chapter, we present an overview of industrial networks and the target protocol of our project (SafetyNETp), thereafter we move to present the available security protocols for securing network traffic and finally discuss our choice. In the third chapter we go through a brief security analysis and we outline the requirements specification and analysis. The fourth chapter is dedicated to the design of our security layer, it structured into two parts, the first part discusses the global design and the second part discusses the detailed design. Along the fifth and the last chapter, we present some implementation details and we test the performance of our solution.

1 General Context

Introduction

In this chapter, we see a brief presentation of the host organism and we present the problematic as well as the objective and the proposed solution.

1.1 Presentation of the Host Organism

The Institute of Reliable Embedded Systems and Communication Electronics (ivESK) at the University of Applied Sciences Offenburg was formed to focus on wired and wireless networks of embedded systems, as well as their interconnection in Cyber-Physical Systems (CPS). ivESK is especially active in designing and implementing of efficient and modular, wired and wireless communications protocols using embedded systems, for instance, 6LoWPAN, wireless M-Bus, and wired M-Bus. Furthermore, the ivESK also designs and implements integrated security architectures for communication solutions using embedded systems. Actually, Those security architectures are mostly based on TLS and DTLS. ivESK not only provides communication security, also, it provides efficient and secure solutions for embedded computing platforms such as Embedded Linux and virtualization. Last but not least, ivESK provides End-to-end security solutions between resource-restricted devices and powerful components.

1.2 Problem Description

During the past 30 years, the main focus in the development of field area networks or, in other words, Industrial networks was on meeting the technical requirements such as real-time constraint and human and machinery safety, resulting in a wide diversity of systems. These systems were mostly developed as closed environments without interfaces to outside, partly also because of missing of network solutions. Thus security never played a prominent role and was not a primary concern in Fieldbus systems [2].

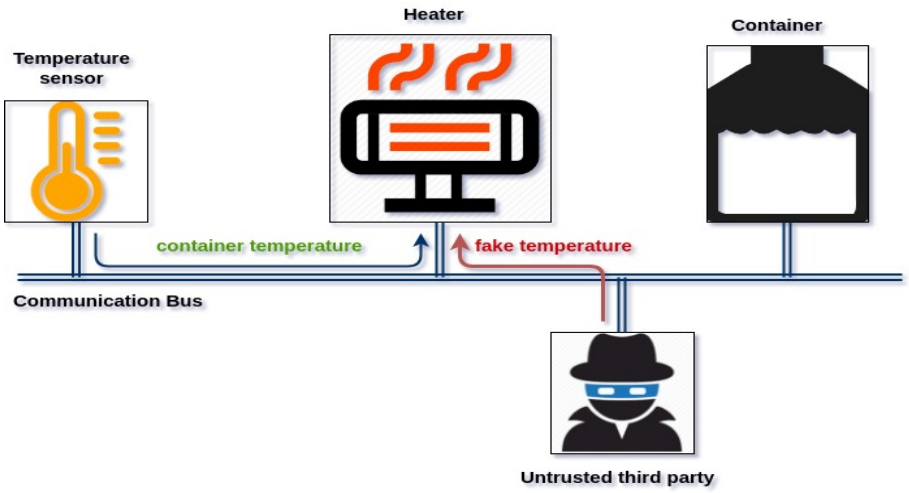

Fig. 1.1: *Lack of authentication exploit*

With the introduction of Ethernet in automation, the usage of TCP/Internet Protocol (IP) became possible, therefore, the same technologies used in LANs and on the Internet are available at the field level. Thus the possibility of being attacked from outside has become possible which was not the case for the Fieldbus systems. However, Fieldbus systems still have security issues because they could be attacked from inside [2].

Security measures for IT systems in any organization aim to achieve three basic security goals, namely, confidentiality, integrity, and authenticity. These measures protect data from unauthorized entities and unauthorized manipulation and ensure availability. In addition, another security goal that is often desired is non-repudiation, which binds an entity to its transacted commitments.

The currently available industrial networks protocols including SafetyNETp, which is the target protocol of this work, do not provide any security measures. SafetyNETp is vulnerable to attacks as it enables unauthorized access in a relatively straightforward manner given that all the communications are performed with no authentication, no integrity check, and no encryption [2].

The Figure 1.1 shows how an industrial network is vulnerable to attacks because of the lack of authentication and integrity checks. This network interconnects a tank, a temperature sensor, and a heater. The temperature sensor sends cyclically the temperature value to the heater. Based on the received value the heater performs an action on the tank (whether to heat, continue heating, or stop). A problem can be introduced if a malicious third party has gained access to the network, this third party can modify the messages being sent by the sensor or send fake temperature values to the heater. Therefore the heater's behavior is affected because it does not have any authentication or integrity check mechanism.

1.3 Objective and Proposed Solution

SafetyNETp is an Industrial Ethernet protocol which provides two communication models, Real Time Frame Network (RTFN) and Real Time Frame Line (RTFL). RTFN and RTFL use different network topologies and operate on different levels in the Open Systems Interconnection (OSI) model. RTFN operates on top of both layer two and four, whereas, RTFL operates only on top of layer two. The next chapter will exhibit the difference between these two variants and their respective communication models. The Figure 1.2 shows the positioning of RTFN and RTFL in the OSI model.

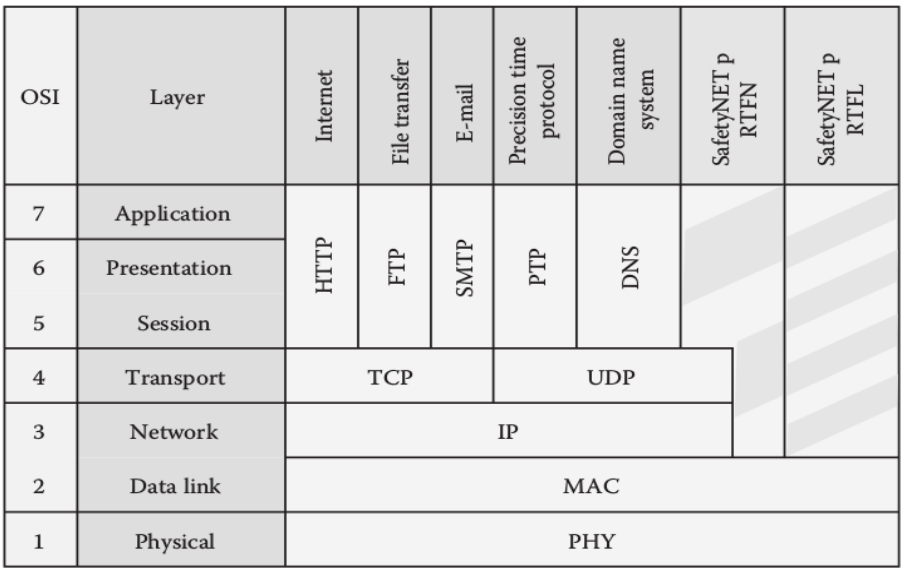

Fig. 1.2: *SafetyNETp in the OSI reference model [5]*

This project aims at securing SafetyNETp RTFN based communication, RTFL being out of the scope of this work; It consists of providing the three basic security services which are confidentiality, integrity, and authenticity. Designing a security layer from scratch is a hard and a very error-prone task. That's why we will be interested in the available solutions which are already tested and assessed by the internet community. TLS, DTLS and IPsec are the most widely deployed protocols for securing network traffic. As cited in the general introduction, TLS is mostly used for securing web traffic and e-mail protocols. TLS can provide confidentiality, integrity, and authenticity by inserting it between the transport and the application layers. However, TLS provides these security services only for reliable transport protocols such as TCP. The incompatibility of TLS with unreliable protocols led to the introduction of DTLS which was introduced to provide TLS's security services for unreliable protocols. Like TLS and DTLS, IPsec provides confidentiality, integrity and authenticity, furthermore, it could be used for securing both reliable and unreliable protocols. In

the next chapter, we study and we choose the suitable protocol to be used. Our solution consists of designing and implementing a security layer for SafetyNETp RTFN based on the chosen security protocol.

Our work will involve the following steps :

- Study SafetyNETp protocol
- Study the available security solutions (TLS, DTLS, IPsec) and choose the suitable one for SafetyNETp
- Find and analyze the existing implementations of the protocol of choice and select the suitable one
- Design the security layer and its integration into SafetyNETp
- Implement and integrate the security layer
- Test the implemented solution

Conclusion

After setting the general context, we move in the following chapter to outline the basic concepts needed for our project and to choose the security protocol to be used

2 State of the art

Using one of the available security protocols, our project aims to secure SafetyNETp RTFN based communication. Therefore an overview and a description of those concepts are required. As SafetyNETp is an Industrial network protocol, we start the chapter with an overview of the history of industrial networks, then we introduce SafetyNETp protocol and its communication models. Finally, we go through an overview of the security protocols to choose the suitable one for our project.

2.1 Industrial Networks

Nowadays we could not imagine a world without emails, online newspapers, blogs, chat and the other services offered by the internet. All these services are made possible thanks to the development of networking. Networking plays an important role not only in our daily lives but also in manufacturing and process automation. Unlike the networks that we are used to in our daily lives, where devices like mobile phones, TVs, printers, and computers are interconnected, in the industry, other types of devices are interconnected such as sensors, motors, actuators, robots and similar other devices. Hence we can speak about industrial networks.

The industrial systems faced the needs for improvement in production monitoring and quality control and at the same time maintaining the costs of all this as low as possible. This happened in the last few decades due to the growth of social needs, which in turn enforce the industrial systems to grow to match up with these needs. So any operation that runs manually had to be replaced with a faster, and more reliable automated operation. This provides the factories with the necessary monitoring which allows better supervisory and quality control. Introducing all this number of automated units into the factories needed an efficient method to connect them together, to communicates with each other, and to transfer the various supervisory data to the monitors.

Ethernet and Fieldbus are the two big keywords when it comes to industrial communication systems. In this section, we go into an overview in which we see the evolution of the industrial networks across the time [2].

2.1.1 Fieldbus Systems

The word Fieldbus is basically the combination of two terms, field and bus. The meaning behind field as defined in the industrial world is a geographical area or space in which the industrial instruments such as sensors and actuators are situated and working. These industrial instruments' tasks are performed at the field level, hence they are commonly called field devices. Regarding the term bus, in computer science, it refers to a line which allows connecting several units and permits to transfer data through it. Before going more into Fieldbus systems, let's see first how was the industrial networks before their introduction [2].

In the industrial world, as shown in Figure 2.1, the first solution that has been used for connecting field devices was star topology [2]. Every field device is wired to a control computer which performs the control and the monitoring tasks. In order to perform these tasks, the control computer needs to transfer the data back and forth from the field devices using the traditional point-to-point technology. This architecture presents several disadvantages, it demands a high cost because of the point-to-point wiring, every field device needs to be wired separately to the control computer, moreover, if this control computer goes down, all the system will collapse.

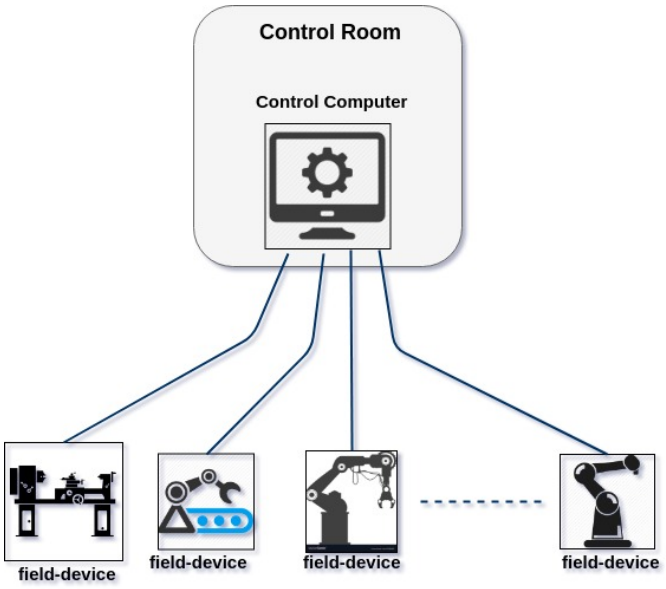

Fig. 2.1: *Centralized point-to-point architecture*

The desire to cope with the wiring problem getting out of hand in large installations was certainly the main impetus for the development of Fieldbus systems. Fieldbus systems were introduced in 1980 and the idea behind them is to connect all the field devices with one single link called Fieldbus [2]. To be able to let all the field devices communicate and transfer data through one single line, a communication protocol should be used to manage the bus access. The communication protocol is

responsible for defining a mechanism for acquiring the bus. Hence the data transferred through the bus will be multiplexed in time. An example of a protocol that can be used for managing the Fieldbus access is the Carrier Sense Multiple Access (CSMA) protocol. The advantages introduced by the Fieldbus systems are many, the most important is the substantial reduction of wiring and adding more flexibility. With Fieldbus systems, the network extension is very much easier to achieve. For instance, CAN is one of the most well-known Fieldbus systems. It is mostly used in automotive networks. The Figure 2.2 shows the architecture of a Fieldbus network.

In today's industrial networks, even systems using topologies other than the linear topology are still called Fieldbus systems.

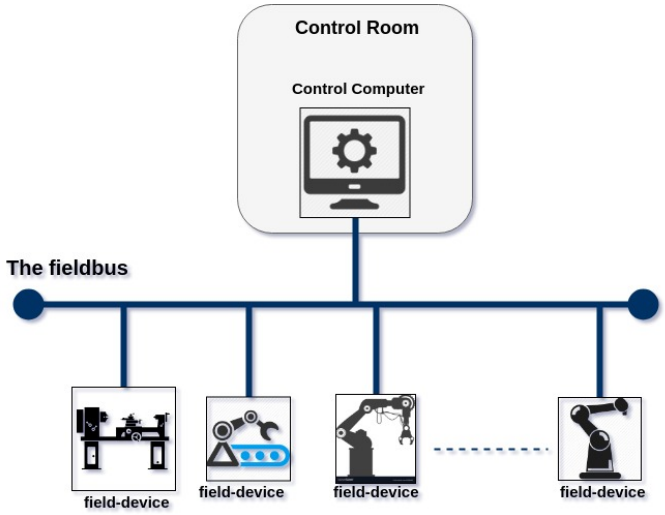

Fig. 2.2: *Fieldbus network*

2.1.2 Industrial Ethernet

Ethernet is a very important communication platform (a physical layer and a data link layer) which was standardized by the Institute of Electrical and Electronics Engineers (IEEE) as IEEE 802.3 [6]. Ethernet is now the most popular and the most widely used network technology in the world which without it, communication between devices in today's modern world would not be possible. For instance, Ethernet is used to connect devices in companies networks like printers and computers, to connect devices in our houses, to connect servers, routers and plenty of other examples.

A key strength of Ethernet is that it enables to run many protocols simultaneously on the same network also it provides flexibility and scalability in a way not seen before. With the growth of Ethernet and having these advantages, the idea of using Ethernet in the industry was introduced. This, in fact, enables a common communication platform for all industrial network protocols. Also, this makes possible to let the field devices communicate with the office devices (using the

TCP/IP model) which provides more flexibility and scalability. Therefore, field devices control and monitoring are not only possible from inside the plant, but also from outside. For instance, a web server could be installed on a field device and could be accessed from an office device to monitor what is happening on the field level [7].

Unfortunately, when Ethernet was introduced, the transmission data rate was 10 Mbit/s which is not suitable for industrial communication. This led to the development of what is called fast-Ethernet with transmission data rates of 100 Mbit/s and 1 Gbit/s which is considered sufficient for industrial usage. Another issue is that using Ethernet in the industry requires additional considerations not seen in Ethernet systems used in an office. In fact devices in industrial environments are exposed to different temperatures, vibrations and other potentially disturbing noises and movements [6].

As we know, in industrial applications, the real-time performance highly matters and has a considerable effect on the automated process. As we discussed previously, using Ethernet in the industry enables the communication between the office devices and the field devices. This is, of course, possible using the TCP/IP communication model. Unfortunately, TCP/IP could not be fast enough to satisfy some applications with tight and critical time requirements. In fact, when data is sent using TCP/IP, like Figure 2.3 shows, the data is packed into the application layer then into the transport layer which basically TCP or UDP then packed into IP datagram and finally into Ethernet frame. When the Ethernet frame is received, the previously cited layers need to be unpacked in order to get the data. The operation of packing and unpacking data through several layers consumes a considerable amount of time that's why alongside using TCP/IP another approach is commonly used in order meet the requirements of critical-time applications. The idea behind this approach is basically to minimize the number of packing and unpacking operations by skipping the network and transport layers (IP and TCP /UDP) which significantly increases the performance. The Figure 2.4 illustrates the second approach [7].

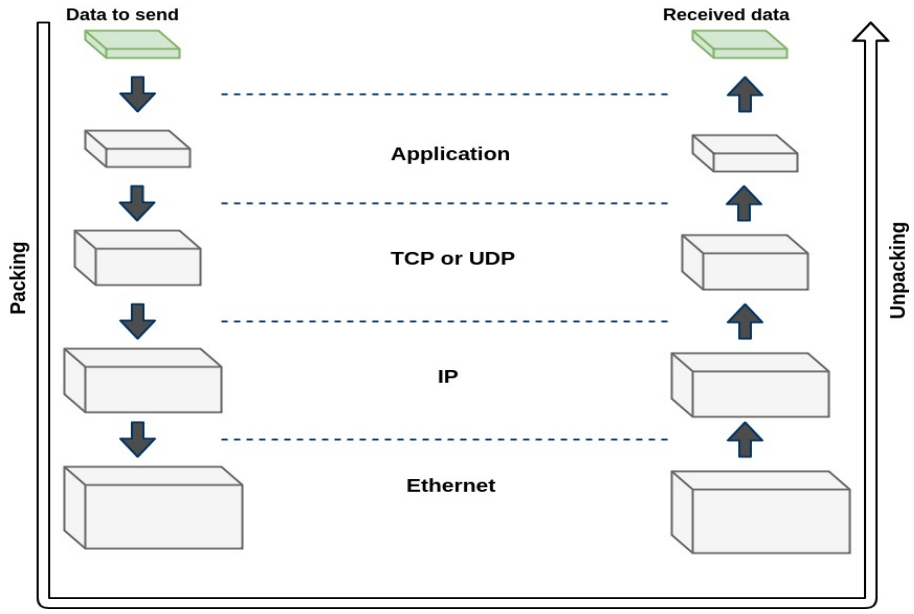

Fig. 2.3: *Data encapsulation for non-critical-time applications*

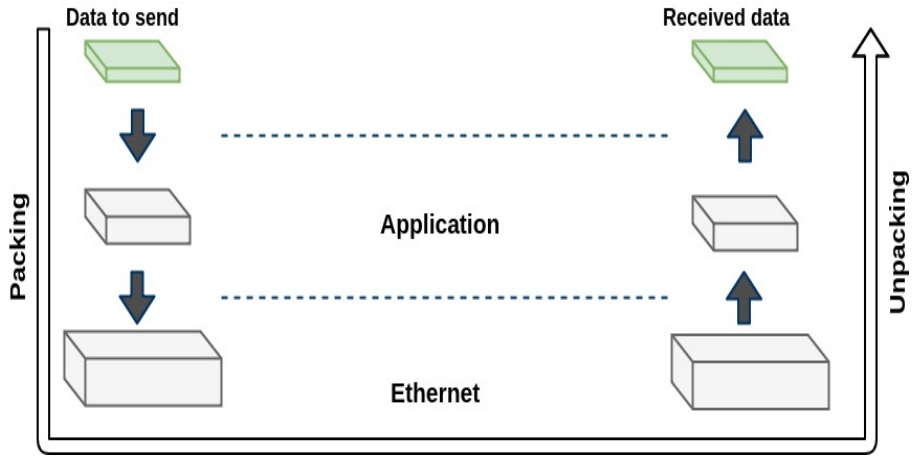

Fig. 2.4: *Data encapsulation for critical-time applications*

2.2 SafetyNETp protocol

SafetyNETp is a communication protocol developed by Pilz GmbH Co. KG, which allows the construction of industrial Ethernet-based networks and is suitable for both real-time communications and safety-oriented applications. The meaning of safety does not refer to networking security, however, it refers to safety applied to humans and to the environment. SafetyNETp is used wherever the temporal and content consistency of communicated data is required to hedge dangers. The dangers can be the endangering of lives as well as the plant assets. The second field of application is the communication of data in real time. With bus cycle times of up to 62.5 microseconds, SafetyNETp can be used even in extremely time-critical areas. Typical fields of application are factory automation (e.g. automobile production), transport technology (e.g. cable cars). In general, all applications of automation and process technology are possible [8].

The protocol enables the transmission of safety-related data on the same cable used for the transfer of control or process data. In fact, it can be used to transfer safety data without the need for a physical dedicated communication line. Combining safe and nonsafe communications over the same medium have several advantages. First of all, the reduced amount of cables needed to connect the network elements makes the wiring of components simpler, thus reducing costs for setup and maintenance. Moreover, the ability to connect safe and nonsafe devices to the same network makes a number of non-safety-related functions (e.g. remote diagnostic, data logging) available for safe devices and at the same time allows standard devices to access safety data [2].

The communication mode commonly used in the industry is the Master/Slave mode. Within this mode, the master can request to read a value from a slave or also write a value to it. The slaves can't communicate directly with each other, they just respond to the read requests sent by a given master and apply the received commands. Hence this mode requires always a centralized controller. The Figure 2.5 shows an example of Master/Slave communication [9].

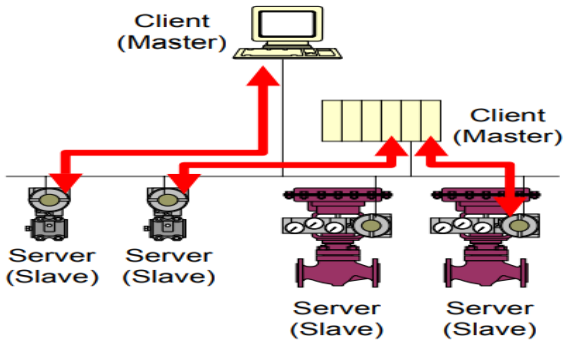

Fig. 2.5: *Master/Slave communication [9]*

In order to avoid the need for a centralized device and to make the data available to all devices, SafetyNETp uses a multi-master bus system where all devices are given the same rights [2]. It uses a Publisher/Subscriber mode, where each device publishes its relevant data and can subscribe, at the same time, the reception of data needed to carry out its tasks, which are produced by other entities in the network. The Figure 2.6 shows an example of Publish/Subscribe communication.

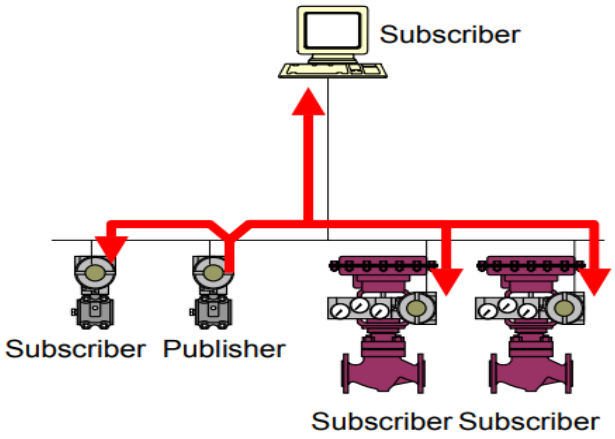

Fig. 2.6: *Publish subscribe communication [9]*

As cited in the first chapter, SafetyNETp provides two communication models RTFN and RTFL. In the next sections, we present the frame layout of both RTFN and RTFL, a brief overview about RTFN and a detailed explanation of RTFN.

2.2.1 Frame layout

As the Figure 1.2 shows, RTFN runs on top of UDP and RTFL runs on top of Ethernet. Therefore RTFN's application data represents UDP's payload and RTFL's application data represents Ethernet's payload. The SafetyNETp header is a one-byte field which indicates the frame type or in other words the payload's content.

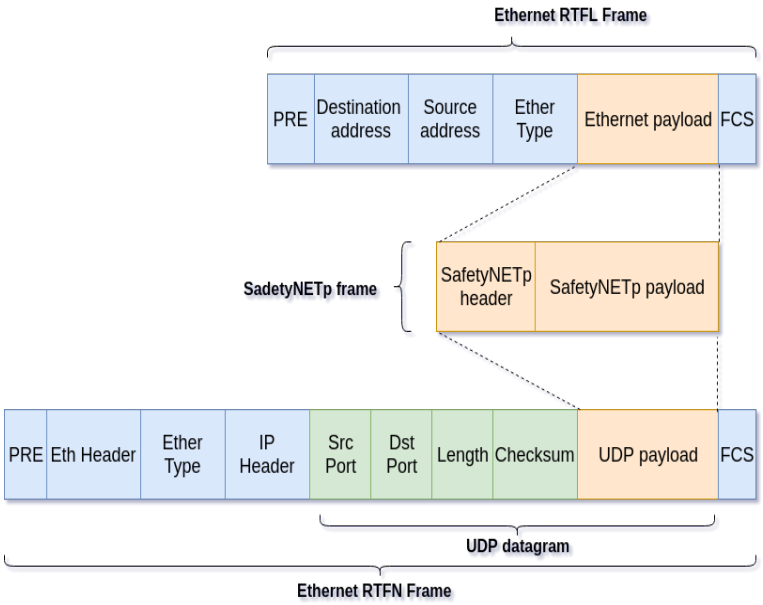

Fig. 2.7: *Mapping of SafetyNETp frame into Ethernet frame and UDP datagram*

2.2.2 RTFL

RTFL has been designed for applications where real-time performance is of primary importance. It is based on a linear topology, and the transport of data is supported through standard Ethernet (OSI layer 2) frames. This communication model enables a minimum bus scan time of 62.5 microsecond and limits jitters to 100 nanoseconds or less, making the protocol suitable for highly demanding control applications [2]. The reasoning behind skipping the network and the transport layers is to increase the performance and this is achieved by reducing the time of packets processing (packing and unpacking). RTFL uses the linear topology which in fact keeps the advantages of Fieldbus systems. All devices are connected through one single line which reduces the wiring costs and makes the data available for all devices.

As shown in Figure 2.8, in an RTFL network, the communication is initiated by a special device called root device (RD). At each communication cycle, the RD creates an RTFL Ethernet frame and sends it along the line. All the cascading devices, also known as ordinary devices (ODs), append their data to the frame. When the frame reaches the end of the line (last device), it is forwarded on the way back to the RD. While traveling in this direction, relevant data are then read by the different ODs. Each RTFL network requires just one RD.

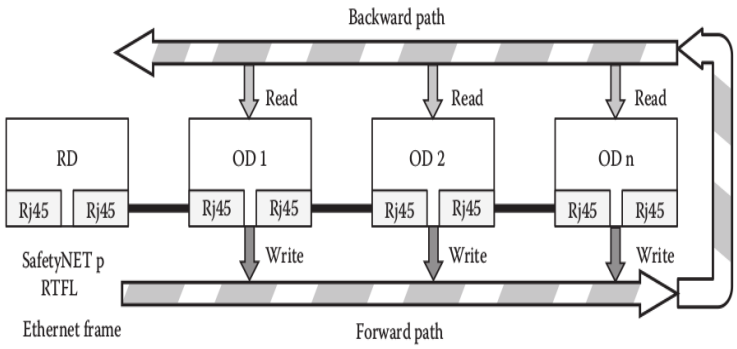

Fig. 2.8: *RTFL physical line*

RTFL is designed to minimize the overhead by using frames as common containers for data from all devices. As depicted in Figure 2.9 a single Ethernet frame can carry all process data needed by the devices. Minimizing the header overhead is important to reach cycle times as short as 62.5 microseconds, which are needed especially in motion control applications.

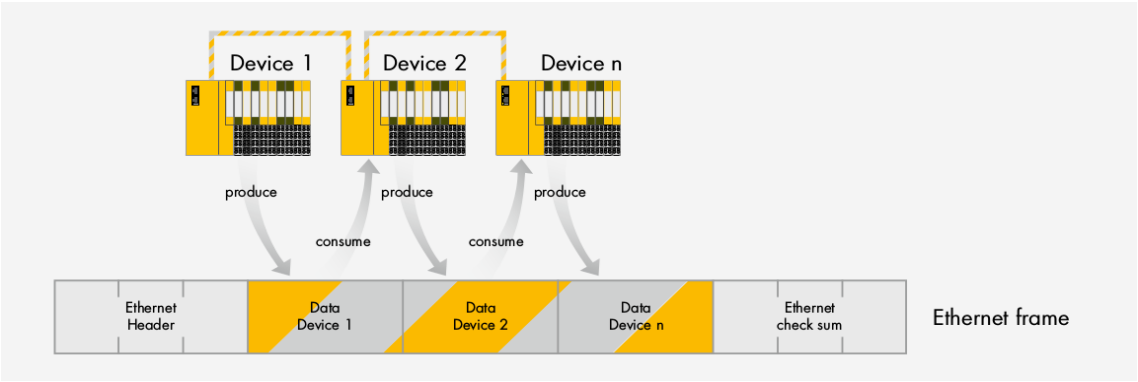

Fig. 2.9: *RTFL Ethernet frame*

2.2.3 RTFN

When real-time requirements are not so tight and a cycle time over 1 millisecond is sufficient, the RTFN communication model can be used to obtain the maximum degree of flexibility and integration with other standard Ethernet-based protocols likely adopted at higher levels of the factory hierarchy. In fact, RTFN can use the standard UDP/IP protocols to transport data, allowing both the routing of frames among different networks and the integration with the existing communication infrastructures. RTFN can be used for communication between field devices and for connecting the field devices to the office network. RTFN is compatible with RTFL and it can be used to connect

multiple RTFL networks together. RTFN does not restrict the usage of a specific topology, it allows using any network topology, for instance, star, tree and ring topologies.

RTFN follows a Publish/Subscribe communication model where the data is exchanged through the concept of topics. An RTFN device could be a publisher, a subscriber or both together. Each publisher has a list of topics. When a publisher receives a *subscribe request* for a topic it has, it responds by sending data about the subscribed topic to the subscriber and this is called publishment. For instance, a temperature sensor could be a publisher which has the temperature as a topic. A device could get the temperature value by subscribing to the corresponding topic. In an RTFN network, as it is shown in Figure 2.10, six types of messages could be exchanged between devices:

- Subscribe request: used by subscribers to request the subscription for a specific topic.
- Subscribe acknowledgment: used by a publisher to confirm that a subscription request has been accepted.
- Unsubscribe request: used by a subscriber to unsubscribe the reception of some topic.
- Still alive: sent by a subscriber to a publisher to inform it that it is still alive.
- Unpublished: sent by a publisher to a subscriber to inform it that no more data will be received for a certain topic.
- Cyclic data: message sent cyclically by the publisher to the subscriber which contains topics' data.

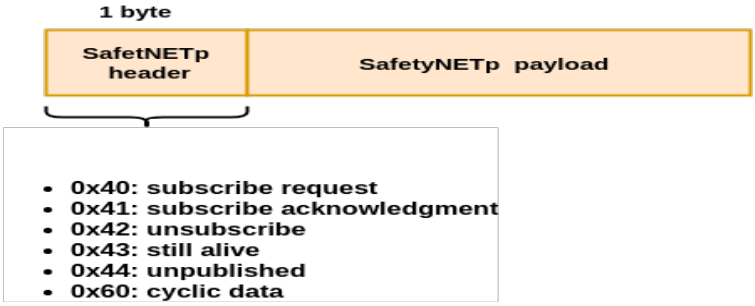

Fig. 2.10: *RTFN frame and messages type*

The sequence diagram shown in Figure 2.11 illustrates the subscription and the publishment process. In this diagram, for simplicity reasons, we assume that the messages are sent reliably (no message loss). The publisher and the subscriber don't know each other in advance, that's why the subscriber starts by sending a *subscribe request* as broadcast (the *subscribe request* will be sent to all the devices in the network). The devices present in the network will receive the *subscribe request* and the device which has the topic available will send back a *subscribe acknowledgment* as unicast to the subscriber. Once the subscription process has finished, the publisher starts publishing cyclically data about the

corresponding topic to the subscriber. The published data is called *cyclic data* and sent only to the subscriber. The subscriber also sends cyclic messages to the publisher. These messages consist of *still alive* messages that are sent in order to inform the publisher that the subscriber is still alive.

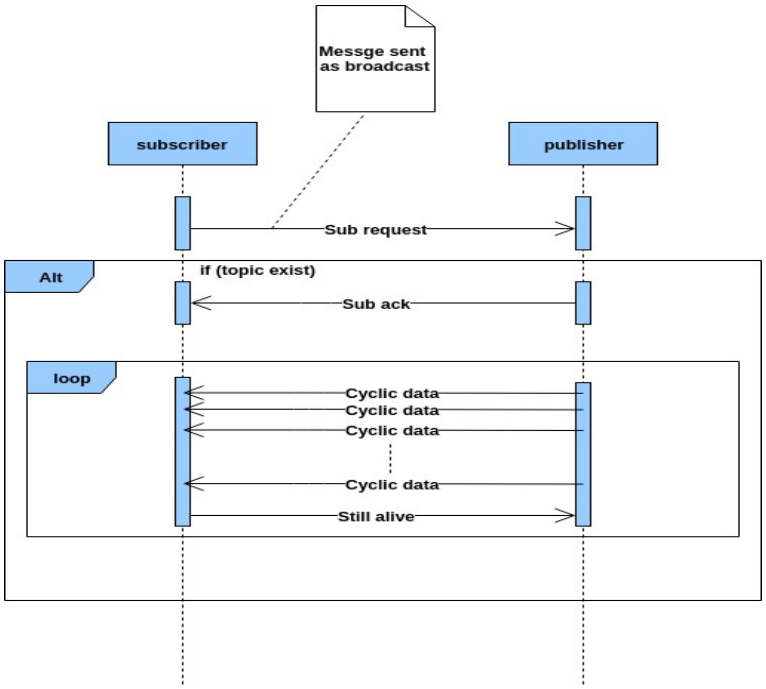

Fig. 2.11: *RTFN Publish/Subscribe communication*

The publisher can stop sending *cyclic data* to the subscriber for multiple reasons. If the subscriber is down, the publisher will not receive anymore *still alive* messages, therefore it will stop sending *cyclic data*. The subscriber can send an *unsubscribe request* to the publisher in order to inform it that the cyclic data for some topic is not needed anymore. The publisher will stop sending *cyclic data* upon receiving the *unsubscribe request*, moreover, the publisher also can send an *unpublished* message to the subscriber to inform it that no more *cyclic data* will be published for a certain topic.

For simplicity reasons, we assumed that there is no message loss. However as we mentioned before, RTFN uses UDP as a transport layer which does not provide reliable transfer. What will happen if for example the *subscribe acknowledgment* sent by the publisher is lost? In fact, the subscriber will keep sending always *subscribe requests* until receiving a *subscribe acknowledgment* from the publisher.

2.3 Security Protocols

In today's internet, almost everything is based on the TCP/IP model. The internet was primarily a small network connecting a community of researchers, but with the success of TCP/IP the internet expanded to become a gigantic global network connecting devices of all types. With the huge success of TCP/IP and the growth of the internet, TCP/IP application became attractive for the industry and the automation area (e.g. Industrial Ethernet). However, no security measures were taken into consideration while designing TCP/IP as it was only for interconnecting small and private networks. In this section, we discuss the most common IP security measures.

2.3.1 Secure Sockets Layer/Transport Layer Security

Secure Sockets Layer (SSL)/TLS is the most widely deployed protocol for securing network traffic. It is widely used for protecting Web traffic and for e-mail protocols such as Internet Message Access Protocol (IMAP) and Post Office Protocol (POP). TLS is developed for any reliable transport protocol which maintains the messages' order. It is typically used for securing TCP applications (e.g. HTTP).

Introduction and History

The first SSL version was introduced by Netscape in 1994. In 1995 and 1996, SSL versions 2.0 and 3.0 were released and then came the release of TLS 1.0 which is, in fact, the same as SSL 3.1. Basically, there is no difference between TLS and SSL, beginning from SSL version 3.1, the protocol name has been changed to become TLS 1.0 [10]. The last standardized TLS version today is 1.2 which was released in 2008 in RFC5246 [3] , moreover, drafts are available for TLS version 1.3. The last published draft of TLS 1.3 was on 20 March 2018 [11].

The goal of SSL/TLS is to provide the three crucial security services:

- Confidentiality: the data is sent encrypted and only the devices which have the keying material could decrypt and read the data. The encryption is performed via symmetric encryption algorithms such as Data Encryption Standard (DES) or Advanced Encryption Standard (AES).
- Integrity: communication partners can detect changes to a message during transmission. this is achieved using Message Authentication Code (MAC).
- Authentication: the recipient of a message can identify its communication partner and can detect if the received message has been forged. The authentication is likely based on a pre-shared key or on client's and server's certificates.

TLS is based on a sub-protocol named record layer protocol which on top of it we can find four protocols, the Handshake protocol, the Change Cipher Spec protocol, the Alert protocol and the Application Data protocol (Figure 2.12). Among these four protocols, the most important one is the Handshake protocol which is used for establishing an authenticated cryptographic secret between two endpoints by commonly using public key cryptography [3].

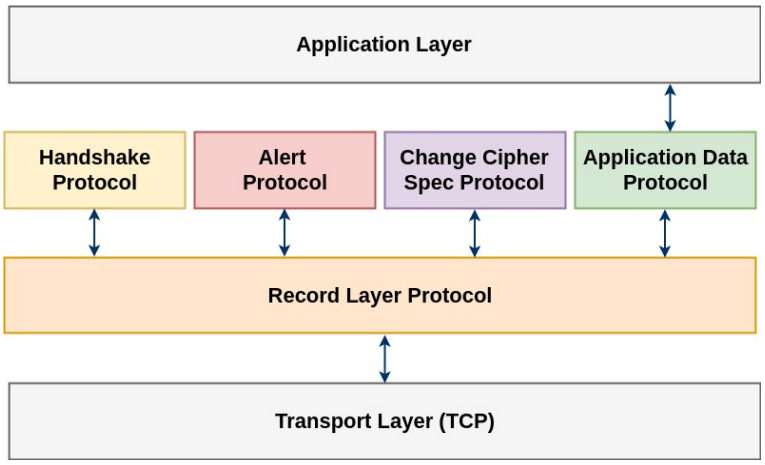

Fig. 2.12: *TLS protocol architecture*

TLS Handshake

TLS is designed to work on top of a reliable transport protocol typically TCP. It follows the Client/Server communication model and its goal is to establish a secure connection between a server and a client and let them exchange data securely across an untrusted network. To establish a secure connection, the TLS handshake needs to take place. During the handshake, important messages are exchanged between the client and the server. With the messages exchanged the client and the server basically agree on the protocol version, select the encryption and the MAC algorithms, authenticate each other by exchanging and validating digital certificates and share a symmetric encryption key to be used for encryption [3].

As standardized in RFC5246 [3], a full SSL/TLS handshake contains the following messages (Figure 2.13):

- Client Hello: it is the first handshake message and it is sent from the client to the server. It contains the supported SSL and TLS versions, a random number, a session id and the supported compression methods and cipher suites. In fact, cipher suites are very important as they define the key exchange algorithm, the authentication algorithm (digital signature algorithm), the data encryption algorithm as well as the MAC algorithm (Figure 2.14).

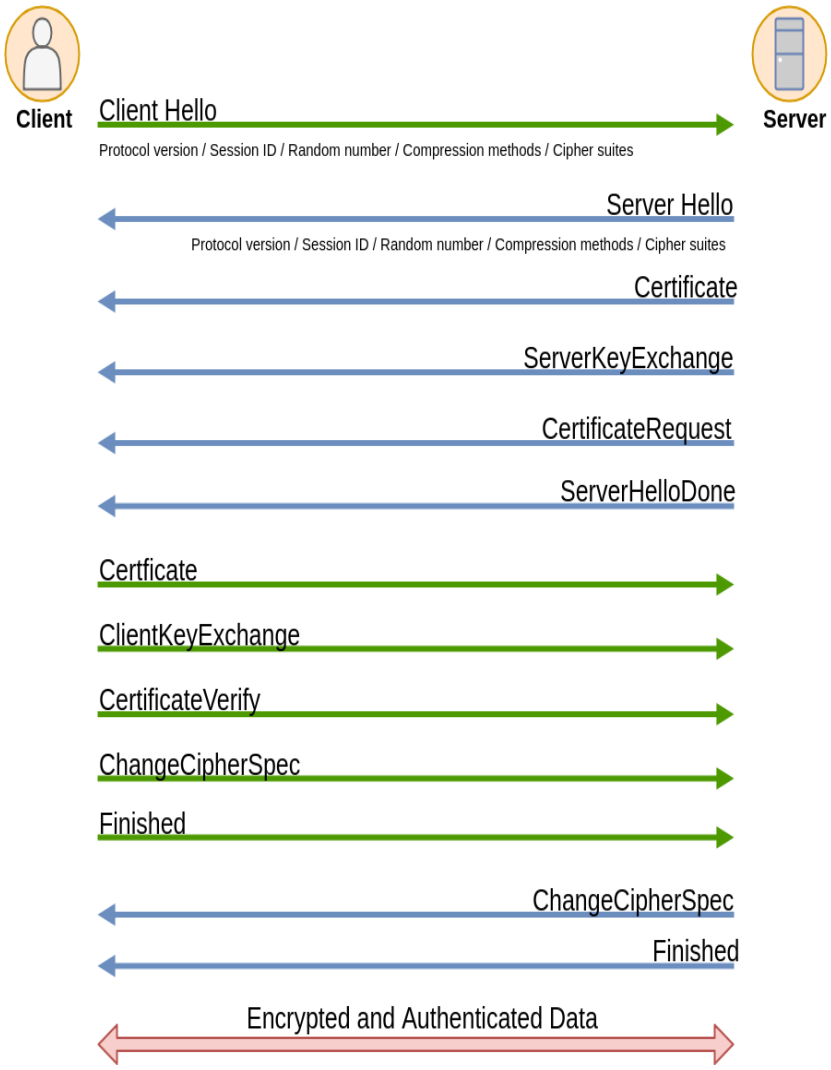

Fig. 2.13: *SSL/TLS full handshake*

Fig. 2.14: *TLS cipher suite example*

- Server Hello: upon receiving the Client Hello, the server sends back the Server Hello message which contains the selected parameters (key exchange algorithm, the authentication algorithm...).
- Server Certificate: it is optional to send, it lets the client verify the identity of the server.
- Server Key Exchange: this message is not always present in the handshake, it is only sent if the server certificate is not sufficient to allow the client to exchange the secret key.
- Certificate Request: it is optional to send, it informs the client to provide its certificate for authentication.
- Server Hello Done: sent by the server to indicate the end of the Server Hello and the associated messages (Server Certificate, Server Key Exchange, and Certificate Request). Upon receiving the Server Hello done, the client verifies the validity of the server certificate and the parameters selected by the server in the Server Hello.
- Client Key Exchange: it is sent by the client after receiving the Server Hello Done to set the Pre-master Secret (PMS). The content of this messages depends on the selected key exchange algorithm, if RSA is chosen, the PMS is encrypted against the server's public key, however, if Diffie-Hellman key exchange (DH) is chosen, DH parameters will be sent. The PMS secret will be used on both sides to compute a shared Master Secret (MS) which will be used for encryption, decryption and MAC calculation.
- Client Certificate: sent if required by the server, it allows the server to verify the identity of the client.
- Certificate Verify: if the client sends a certificate with signing ability, a digitally-signed Certificate Verify message is sent in order to explicitly verify the certificate.
- Change Cipher Spec: sent by both the client and the server. This message indicates that the following messages will be sent encrypted using the shared keying material.
- Finished: sent by both the client and the server after the Change Cipher Spec message to verify that the key exchange and authentication processes were successful. It is the last handshake message and the first encrypted one.

How the security services are ensured?

In our project, when we say security services, we mean basically confidentiality, integrity, and authenticity. When the server sends its certificate, the client verifies the validity of the certificate using the public key of the Certificate Authority (CA). If the certificate is valid, the client generates a PMS and sends it to the server. Not only the client can verify the server identity, also the server verifies the client identity, in the same way, using the client certificate. If the client certificate is valid the server will accept the provided PMS. Using the PMS, both the client and server derive a MS which is in turn used to derive other several keys. These keys are used for exchanging authenticated and encrypted data with the capability to the check its integrity. The confidentiality is ensured by encrypting the data, the authentication and the integrity are both ensured using a MAC algorithm [3].

2.3.2 Datagram Transport Layer Security

A lot of applications today in several domains are delay sensitive and have tight time constraints. For instance, we can speak of industrial communication, IoT, internet telephony, online gaming and several other applications. Due to the time constraints, for this kind of applications UDP is used instead of TCP .

Introduction and History

Application layer protocols that work on top of TCP such as Hypertext Transfer Protocol (HTTP), Secure Shell (SSH) or File Transfer Protocol (FTP) can be easily secured using TLS as it is designed to secure applications working on top of a reliable transport protocol. Unfortunately, for datagram based applications, no such alternative exists. Therefore, similarly to TLS, DTLS was introduced to allow the use of the same security services provided by TLS but for datagram based applications.

Over the years, TLS has become more robust and has been refined to withstand numerous attacks that's why DTLS was designed to be as similar as possible to it. By minimizing the changes, the risk of introducing new weaknesses or vulnerabilities will be reduced. Moreover, DTLS implementation will be easier by reusing TLS pre-existing infrastructure [12].

The last standardized DTLS version is 1.2, it was standardized in 2012 as RFC6347 [4]. Like TLS, drafts have been published for DTLS version 1.3. The last draft was published on 02 July 2018 [13].

DTLS Design

TCP provides a reliable bi-directional tunnel for bytes, where all bytes eventually reach the receiver in the same order as what the sender used to send them. TCP achieves that through a complex assembly of acknowledgment messages, transmission timeouts and retransmission. As TLS uses TCP, it does not encounter issues related to packet loss and reordering which is not the case for DTLS.

The unreliability of datagram protocols creates problems for TLS in two levels. First, the messages decryption is dependent from the previous messages as the integrity check is based on the implicit sequence number. If record N is not received, then the integrity check on record N+1 will be based on the wrong sequence number and thus will fail. Second, TLS assumes that the handshake messages are sent reliably, hence, the connection cannot be established if a handshake message is lost [3].

In order to handle packet loss, DTLS uses a retransmission timer described in details in RFC6347 [4]. DTLS includes an explicit sequence number which is used for MAC calculation. It is used also to detect replay attacks and reordering. Furthermore, a new field was introduced named epoch which is incremented each time a CipherChangeSpec message is sent.

Some TLS cipher suites retain a cryptographic context between records which does not allow to process individual records. This will cause problems with unreliable protocols, therefore, DTLS banned the usage of this kind of cipher suites [3].

TLS and DTLS handshakes are the same except that DTLS has introduced a new message which does not exist in TLS's handshake. The new message is named Hello Verify Request, it contains a cookie and it is sent back to the client just after receiving the Client Hello message. the client should send back another Client Hello which contains the same cookie. If the cookie is verified the handshake will continue (same TLS handshake steps), otherwise, the server will close the connection. The cookie mechanism allows reducing the risk of Denial of Service (DoS) attacks. No such message exists in TLS handshake because the TCP handshake is established before starting the TLS handshake which will make the DoS attack difficult. This is not the case for datagram protocols because no connection is established prior to the DTLS handshake, hence, the cookie mechanism is needed [4].

2.3.3 Internet Protocol Security

IPsec is a protocol suite which was introduced to secure network traffic and more specifically IP and its upper layers protocols, typically TCP or UDP. IPsec is situated in layer three of the OSI

model and it offers data integrity, replay protection, authentication, and confidentiality natively within the IP layer. IPsec offers two communication modes, tunnel mode and transport mode (Figure 2.15). The tunnel mode provides network-to-network security whereas the transport mode provides host-to-host security. In fact, in the tunnel mode, the sender sends the message as it is (unprotected) to the gateway, then the gateway encapsulates the message and sends it securely. The receiving gateway decapsulates the secured message and passes the original message to the receiver if the message is verified correctly (e.g. integrity, authentication). In the transport mode, unlike the tunnel mode, protecting the messages is no longer performed by the gateway, instead, the end hosts manage themselves to secure and verify the messages.

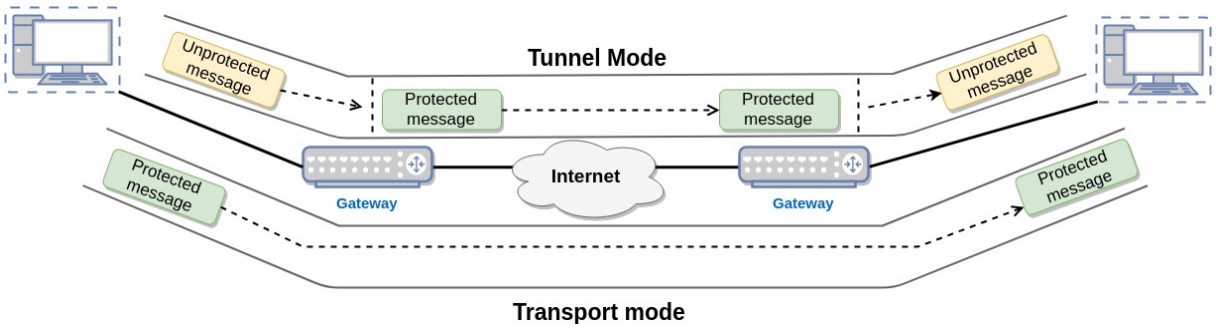

Fig. 2.15: *IPsec communication modes*

To provide its security services, IPsec use three independent protcols:

- Authentication Header (AH): It provides integrity and authentication for the IP header and its payload in both tunnel and transport modes and optionally provides protection against replay attacks. However, it does not provide confidentiality. The IP header mutable fields (the fields which can change during the transmission) like Time To Live (TTL) are not authenticated.
- Encapsulating Security Protocol (ESP): It provides confidentiality, integrity, and authentication for the IP payload in both modes. However, it does not provide protection for the IP header.
- Internet Key Exchange (IKE): It allows to share an authenticated secret key to be used for encryption, authentication, and integrity checks. it uses DH as a key exchange algorithm and RSA for digital signatures.

As depicted in Figure 2.16, whether AH or ESP is adopted, if the tunnel mode is used, when the gateway receives the original IP datagram from the sender, it encapsulates it into a new IP datagram containing a new IP address which corresponds to the receiving gateway and then the message is protected. When the transport mode is used the original IP datagram is protected by the sender and sent without new IP encapsulation. ESP and AH could be used separately or together according to the needed security services.

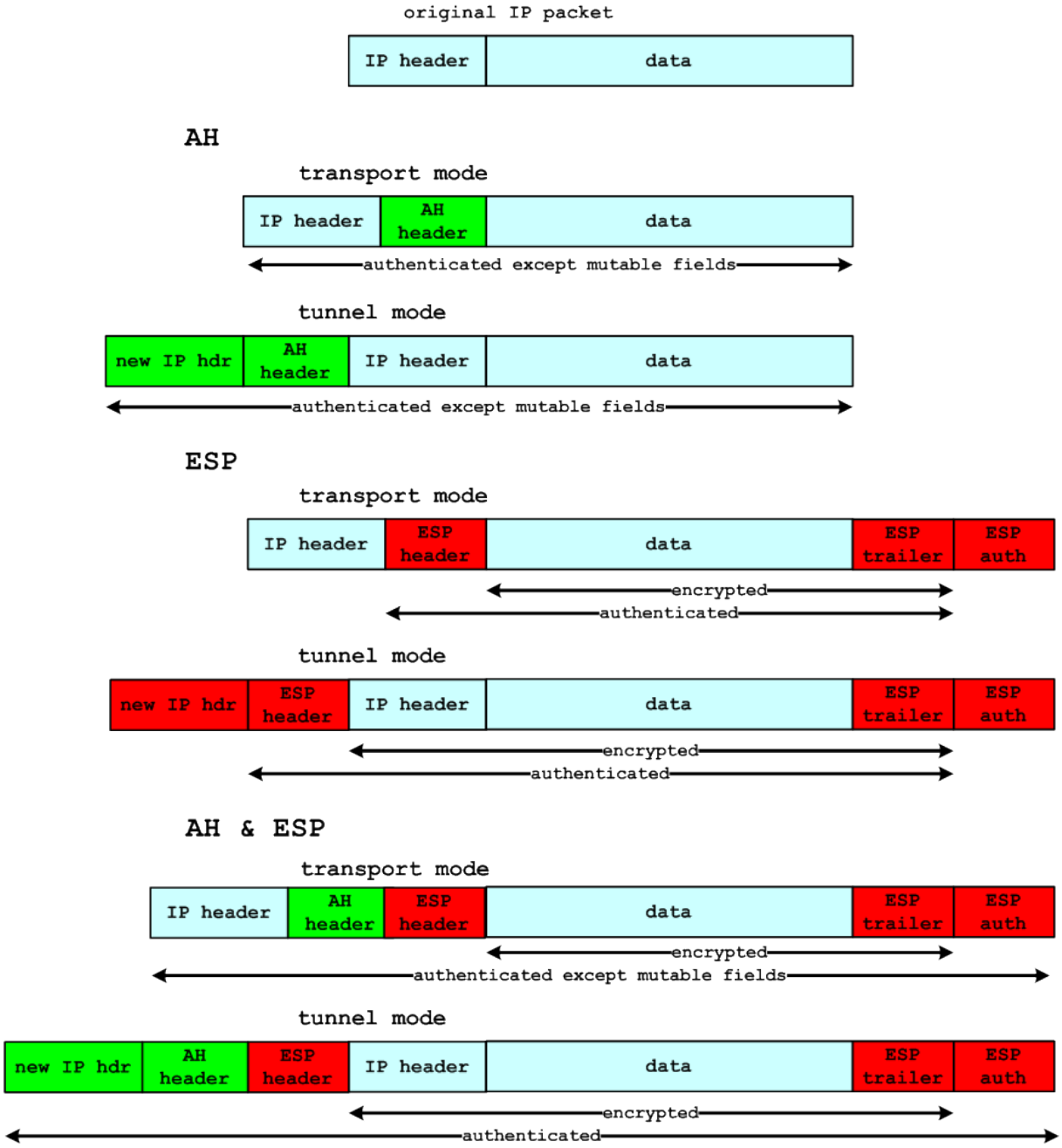

Fig. 2.16: *IPsec protocols*

2.4 Choice of Security Protocol

We need to take into consideration first that SafetyNETp RTFN works on top of UDP. DTLS and TLS provide the same security services. TLS is used for reliable transport protocols whereas DTLS is for unreliable ones. IPsec can be used for securing any type of traffic working on top IP. Therefore our choice is restricted between DTLS and IPsec.

Automation systems and their real-time operating systems often are resource constrained and do not include typical security technologies, and there may not be available computing resources to retrofit these security technologies. IPsec tunnel mode is a solution as the cryptographic operations are performed by the gateways. The end devices send and receive the messages transparently, no additional treatment is needed for them. However, the tunnel mode protects the network only from outside because messages exchanged on the local network are not secured by the gateway. Hence, we could have serious problems if an attacker gains access to the local network. IPsec transport mode provides end-to-end security and allows to secure the network both from inside and outside, however, the cryptographic operations are done by the end devices which have constrained resources. IPsec operates at the IP layer, it generally must be implemented in the operating system kernel, either directly compiled in or linked in as a loadable module. This makes IPsec fairly inconvenient to install on non-IPsec systems. AH and ESP are typically implemented in the kernel as part of the IP stack, while IKE is implemented as a user daemon. This makes its integration difficult as most of the embedded IP stacks do not implement IPsec.

DTLS provides end-to-end security and like IPsec, the cryptographic operations are performed by the end devices. DTLS operates between the transport layer and the application layer. Using DTLS, it is easy to secure an application protocol by simply inserting it between the application and the transport layers. DTLS is IP-stack-independent which means that it does not include any modification to the underlying IP stack and could be used with any IP stack implementation, hence, its integration in embedded IP stacks is a lot easier.

Having that Both, IPsec and DTLS provide the needed security services (confidentiality, integrity, and authentication), and based on the previous paragraphs, we decide to use DTLS.

Conclusion

After going through the necessary concepts needed for our project, in the next chapter, we go through a brief security analysis and we present the functional and non-functional requirements.

3 Requirement Analysis and Specification

After introducing the basic concepts, we move in this chapter to conduct first a brief security analysis in which we outline the assets as well as the threats, thereafter, we go through the requirements analysis in which we cite the functional and the non-functional requirements. Finally, we present the requirements specification using a use case diagram.

3.1 Threat Analysis

Security is the set of means implemented for minimizing the vulnerabilities of an information system against accidental and/or intentional threats. Its main goal is to protect the system sensitive elements and resources from threats according to the needs. Therefore, we need to define as a first step what are the assets that we will be protecting, as a second step we present the threats related to the scope of our project.

In our project, our goal is to provide an end-to-end security for SafetyNETp RTFN based network, therefore, the assets we will be protecting are every device in the network which runs SafetyNETp RTFN application. In today's SafetyNETp RTFN based networks, data transmission is performed without any special security measures. The lack of security measures can create unforeseen security threats to the network, the data, and the resources. In our project, we are interested only with threats with regards to the general security objectives namely confidentiality, integrity, and authenticity. Any other threats are out of the scope of our project, for instance, physical attacks, vandalism, theft of equipment, etc.

In the following, we highlight some of the threats that a SafetyNETp network could face because of the lack of security measures.

- Identity impersonation: To get *cyclic data* about a certain topic, subscribers send *subscribe requests* as broadcast. Upon receiving *subscribe acknowledgments*, subscribers start consuming *cyclic data* without any verification about the publishers' identities. Therefore, an untrusted third party can easily impersonate the identity of a device to create and send its own malicious *cyclic data*. As no authentication mechanism is present, subscribers take decisions about the action to perform based on malicious and untrusted *cyclic data* which is highly dangerous. Furthermore, an attacker can easily flood a publisher with a huge number of *subscribe requests*

using spoofed IP addresses, hence, overloads the publisher and the bandwidth and eventually can cause a DoS attack.

- Data tampering: When a *cyclic data* message is received, no integrity checks are performed by the subscriber and the data is consumed directly. In fact, SafetyNETp RTFN application layer does not provide any mechanism for verifying messages' integrity. Therefore, with a man in the middle attack, an attacker can easily intercept the messages being exchanged and modify the *cyclic data* being sent from a publisher to a subscriber.

- Eavesdropping: The data communicated between two plants in different geographical areas passes through the internet. SafetyNETp RTFN application layer does not provide any mechanism for protecting confidential data, in fact, the messages are sent as plaintext. Without encryption, exchanging data through the internet, any third party can easily have access to highly confidential data.

To illustrate better, we take a real-life example where SafetyNETp is used and where security is very important. In fact, SafetyNETp provides a solution for railways automation [14] where SafetyNETp devices control the trains' tracks, the level crossing protection system as well as functions in and around the trains. Having snow and low temperature could interrupt the railway's operations, that's why Pilz hand over a heating system to provide the needed temperature. This is implemented by a temperature sensor and a heater. The heater gets the temperature value from the sensor and takes the decision based on that. Figure 3.1 shows the railway after and before that the heater took the decision. With the lack of authenticity, an attacker can send false temperature values to the heater by impersonating the identity of the sensor. The attacker can keep sending high temperature values that keeps the heater on an idle state and therefore the railway become unusable. An attacker is able also with a man in the middle attack to change the values being sent by the sensor. As a result, the heater takes decisions based on unauthenticated or tampered data. SafetyNETp provides a remote access to the devices for control and diagnostic purposes. Exchanging data through the internet allow third parties to gain access to confidential data like trains' and railways' states.

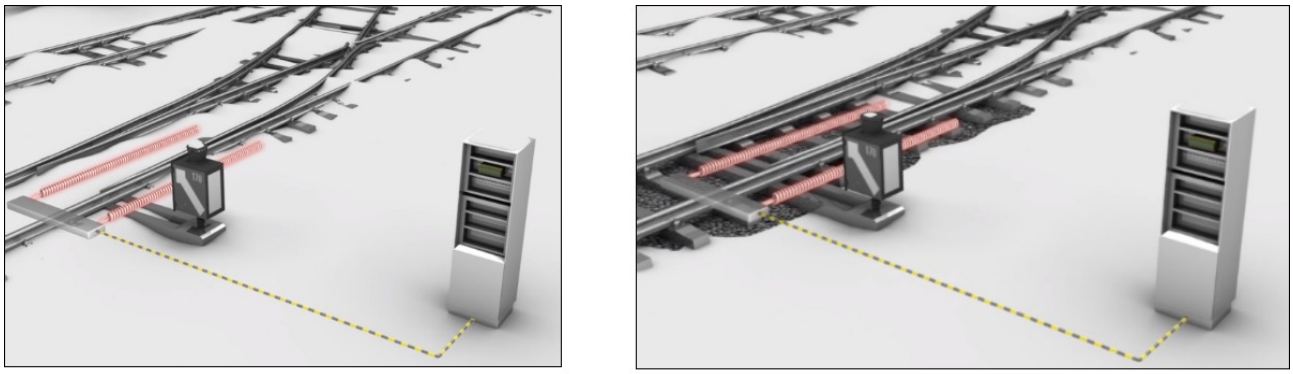

Fig. 3.1: *Pilz railway heating system[14]*

3.2 Requirement Analysis

In this section, we present the functional and the non-functional requirements that our security layer should satisfy.

3.2.1 Functional Requirements

Based on DTLS, our security layer should provide the following requirements:

- Authentication: Recipients of a message can identify their communication partners and can detect if the sender information has been forged.
- Integrity: Communication partners can detect changes to a message during transmission.
- Confidentiality (Optional): If the messages are confidential, they could be encrypted and only the authorized devices could be able to decrypt messages and read the content.

3.2.2 Non-functional Requirements

Our security layer should respect the following constraints:

- Transparent integration: The security layer should not perform any changes to SafetyNETp application layer state machine.

- Easily Configurable: Changing the parameters of the security layer should be easy for SafeyNETp developers.

- Time and memory constraints: The security layer needs to keep SafetyNETp cycle time as it is as much as possible and to optimize memory consumption.

- Maintainability: The security layer needs to be easy to understand and to maintain.

3.3 Requirement Specification

In this section, using a use case diagram, we formalize our requirements by illustrating the interaction between the security layer and our only actor which consists of SafetyNETp application layer.

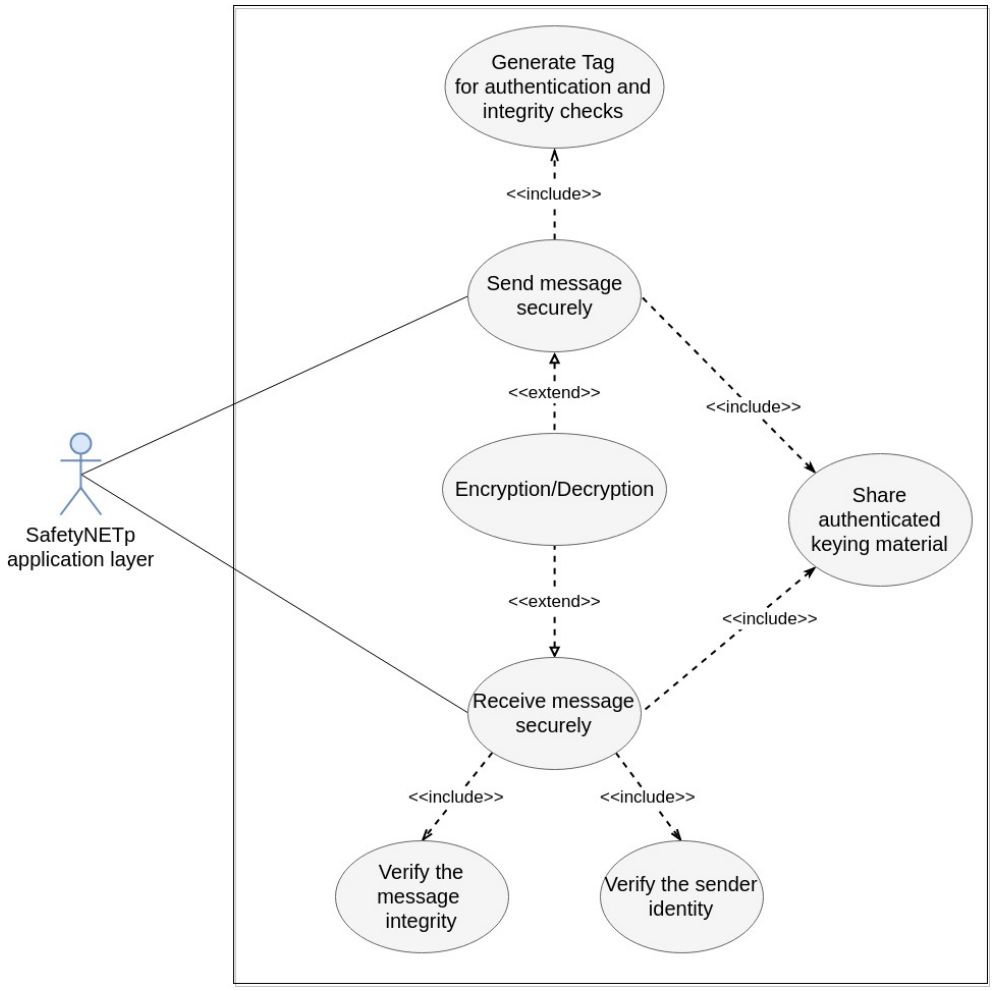

Fig. 3.2: *General use case diagram*

As depicted in Figure 3.2, our security layer allows SafetyNETp application layer to both, send and receive messages securely. Indeed, we need first the establishment of shared authenticated keying material. Thereafter, if the operation is sending, a tag will be generated by our security layer and sent to allow the receiver to authenticate the sender and verify the integrity of the received message. If the operation is receiving, using the tag, the receiver is able to authenticate the sender and verify the message integrity. In some application we may don't need confidentiality, therefore, the encryption is optional. In fact, in applications where confidentiality is not required, sending data without encryption allows reducing significantly the CPU usage.

Conclusion

After we set the requirements that our security layer should satisfy, we pass in the next chapter to discuss the global and the detailed design.

4 Design

In this chapter, we provide and discuss the candidate solutions for integrating the DTLS protocol within SafetyNETp RTFN communication model. Our security layer DTLS-based design is discussed in two parts. In the first part, we see the global design in which we discuss the handshake timing, DTLS and SafetyNETp role mapping as well as the secure channel creation. The second part is the detailed design, in this section, we go deeper into the solutions to see the security layer positioning, how the outgoing and incoming message are treated, the frame layout as well as several other details.

4.1 Global Design

As we mentioned before it is easy to provide a secure channel for an application layer protocol by inserting DTLS or TLS (based on the underlying transport layer) between the application layer and the transport layer. For instance, HTTPS is a secure version of HTTP, this security is provided by inserting TLS between TCP and HTTP. This is made easy due to the fact that HTTP and TLS follow both the Client/Server communication model.

Unlike HTTP and similar protocols, providing a DTLS-based security layer for SafetyNETp is not as simple as it seems. In fact, SafetyNETp is based on Publish/Subscribe communication model, on the other hand, DTLS is designed for Client/Server based applications. Hence, we need a solution which allows mapping and adapting DTLS to SafetyNETp. Furthermore, an important question could be asked: in which step of SafetyNETp communication the handshake should be established?

In this section, we see when the DTLS handshake is established, how to manage mapping the roles for the different communication model and we present how a secure channel is provided.

4.1.1 When To Establish The Handshake?

As shown in Figure 4.1 there are three possible alternatives. Establish the handshake before sending a *subscribe request*, after sending a *subscribe request* but before receiving an acknowledgment, and the last alternative after receiving an acknowledgment. These cases correspond to the device which

takes the subscriber role. Regarding the publisher, there are two alternatives, whether after or before sending a *subscribe acknowledgment*.

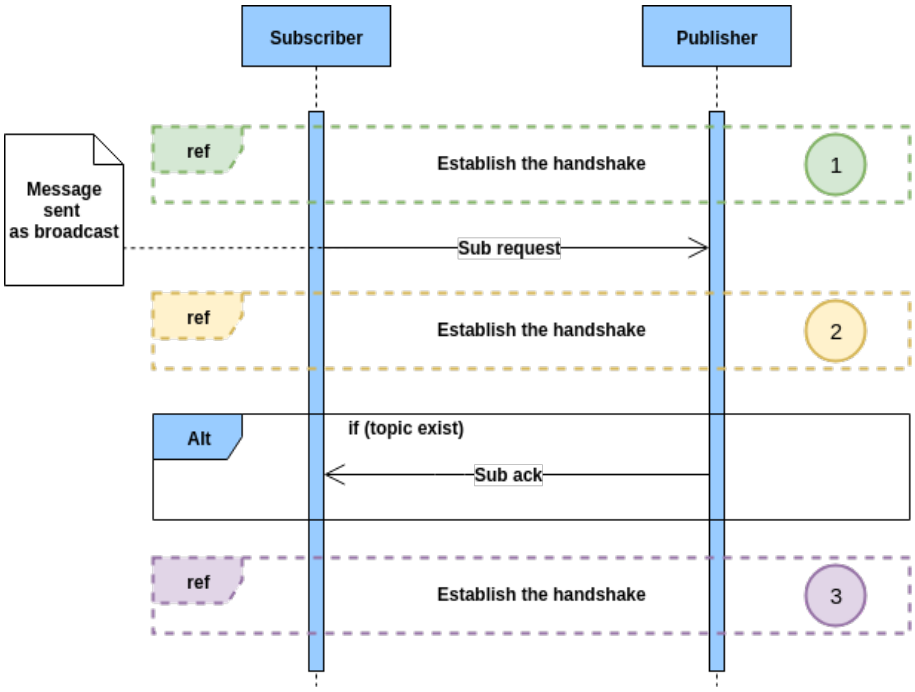

Fig. 4.1: *Possible cases of handshake establishement*

As we mentioned before publishers and subscribers don't know each other in advance. Therefore the subscriber could not establish the handshake in the first and the second cases because the publisher which is the second handshake end-point is unknown at this time. For the first two cases, the solution is to establish the handshake with all the devices. The handshakes establishment is triggered by the devices' startup, therefore, it is not possible to add a new device to the network at runtime. Assuming that we have a network of RTFN devices, at startup time, all the devices will establish the handshake with each other and they start exchanging data. If we add a new device after this phase, no handshake will be established with it as the handshakes phase has already finished, hence, the new device will not be able to communicate securely. To be able to add a new device to the network, the devices should be restarted. Furthermore, this solution is not efficient in term of memory usage and startup time. For each handshake, memory resources need to be allocated which is memory waste because not all the channels will be used by the subscriber. Besides, the startup time will increase if the number of devices is important. This method introduces another inconvenient, a session between a publisher and a subscriber should remain always open even if no communication is needed. In fact, once the session is closed, they could not establish a new session as it is triggered by the startup. Letting unnecessary sessions open leads to memory waste.

Regarding the third case, after that the subscriber receives a *subscribe acknowledgment* from the publisher, at this level, the subscriber knows the second handshake end-point and the handshake could be established. The disadvantages of the previous solution are not present here. Only necessary channels will be created which optimizes the memory usage and reduces the startup time. In addition unlike the previous cases, unnecessary sessions could be closed as the handshake is triggered by the subscription process (more details about sessions closing are given in the next sections). Unfortunately, our security layer needs in this case to inspect the application layer header in order to take the decision. Furthermore, the publisher starts sending *cyclic data* to the subscriber after sending the *subscribe acknowledgment* which presents an issue. In fact the *cyclic data* will be sent while the handshake is being established, however, the identity of the second end-point is not verified. Hence *cyclic data* could be sent to untrusted devices.

The Table 4.1 shows the main differences between the cited alternatives.

	First and second cases	Third case
Memory usage	more	less
Startup time	more	less
Message inspection	no	yes
Flexibility	no	yes
Close unnecessary sessions	no	yes
Cylic data sent while establishing the handshake	no	yes

Tab. 4.1: *Alternatives comparison*

In the third case, the subscriber and the publisher know each other addresses which solves the problems present in the first and the second cases. Comparing these alternatives, the third is better in term of flexibility, memory usage, and startup time. Therefore we choose to establish the handshake after the first subscription process. We see in the next sections how to deal with the *cyclic data* sent while the handshake is being established.

4.1.2 SafetyNETp and DTLS Role Mapping

After choosing when to establish the handshake, an important question could be asked. How to map DTLS roles (client/server) with SafetyNETp roles (publisher/subscriber)? The intuitive solution is that the subscriber takes the role of the client and the publisher takes the role of the server. After sending the *subscribe acknowledgment*, the publisher activates its servers and waits for a

Client Hello message. After receiving the subscribe acknowledgment the subscriber takes the client role and sends a Client Hello to the publisher which has already activated its server and which is ready to establish the handshake. Once the handshake has finished successfully, the publisher can send *cyclic data* securely through the created secure channel.

Unfortunately, if we consider that two SafetyNETp devices could have a mutual subscription, an issue is presented. As shown in Figure 4.2, the devices A and B subscribe mutually to each other in parallel at the same time. As discussed in the previous section the event which triggers the handshake establishment is the subscription process. With the method presented above, having simultaneous subscription will lead to create two channels between A and B. We will end up having a channel for each relation of (publisher,subscriber). In the case shown in Figure 4.2 we have two relations, the relation (A,B), in which A is the publisher and B is the subscriber and the (B,A) relation which is the opposite. Therefore two separate channels will be created between A and B whereas one single channel is sufficient.

Creating more than one channel for the same couple leads to more memory usage that's why we need a mechanism to avoid duplicated channels. After that both of A and B had sent the *subscribe acknowledgment*, they know each other's IP addresses. Based on the IP addresses, each device can choose a distinct DTLS role. The device which has the greater IP address takes the server role, the other device takes the client role. Therefore, we choose to use IP addresses as a criterion to map DTLS and SafetyNETp roles. Other alternatives than the IP addresses could be used like MAC addresses for example. What is important is to make the choice and break the conflict to avoid duplicated channels which could be achieved using this method.

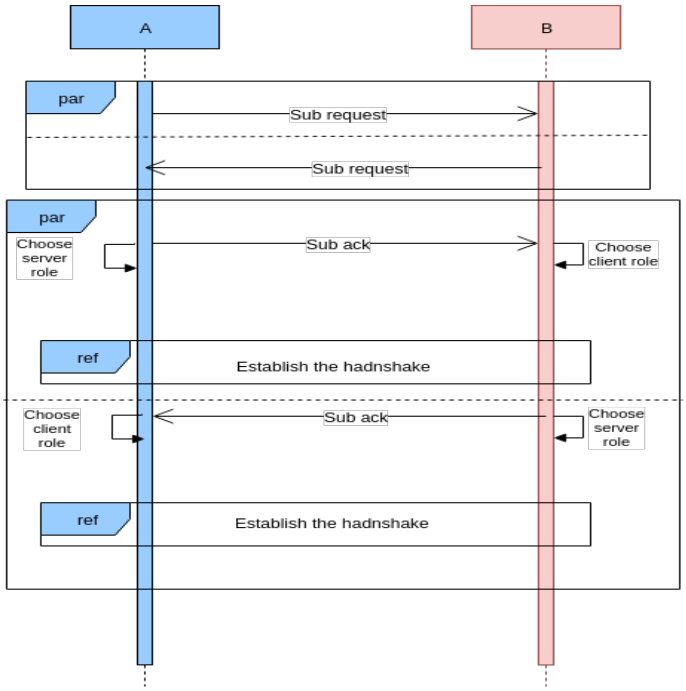

Fig. 4.2: *Simultaneous mutual subscription*

4.1.3 Secure Communication

As a first step, we discussed when to establish the handshake, as a second step we defined a mechanism to map the client/server role with the publisher/subscriber role. As mentioned before, if the handshake is established after the subscription process, the publisher starts sending *cyclic data* while the handshake is being established. This can lead to send *cyclic data* to unauthorized devices as the data is sent before finishing the handshake. In this section, as a next step, we take into consideration this issue and then we see a nominal case of a secure communication between two devices

Cyclic data messages are very important that's why they should not be sent or received from an untrusted device. While the handshake is being established, for the publisher, the identity of the subscriber is not verified yet. As mentioned in the non-functional requirements, our security layer should not introduce any changes to SafetyNETp state machine, therefore, two decisions could be taken, whether to drop or to store the *cyclic data* until finishing the handshake. If the *cyclic data* is stored, once the handshake is finished and the identity is verified, nothing is lost and the *cyclic data* could be sent securely through the secure channel. However, storing the cyclic data can eventually cause a DoS attack. In fact, an attacker could send a huge number of *subscribe request* to a publisher with spoofed IP addresses. The publisher will send a *subscribe acknowledgment* for each received request and then start storing the *cyclic data* until verifying the identities. Eventually, the identities

will not be verified but the publisher memory will become saturated and the publisher will be out of service. This is the first reason for which the *cyclic data* should not be stored, furthermore, the *cyclic data* represent information about a certain topic and based on its content subscribers take actions. Having old information about a topic can cause serious problems, that's why we choose to drop *cyclic data* while establishing the handshake.

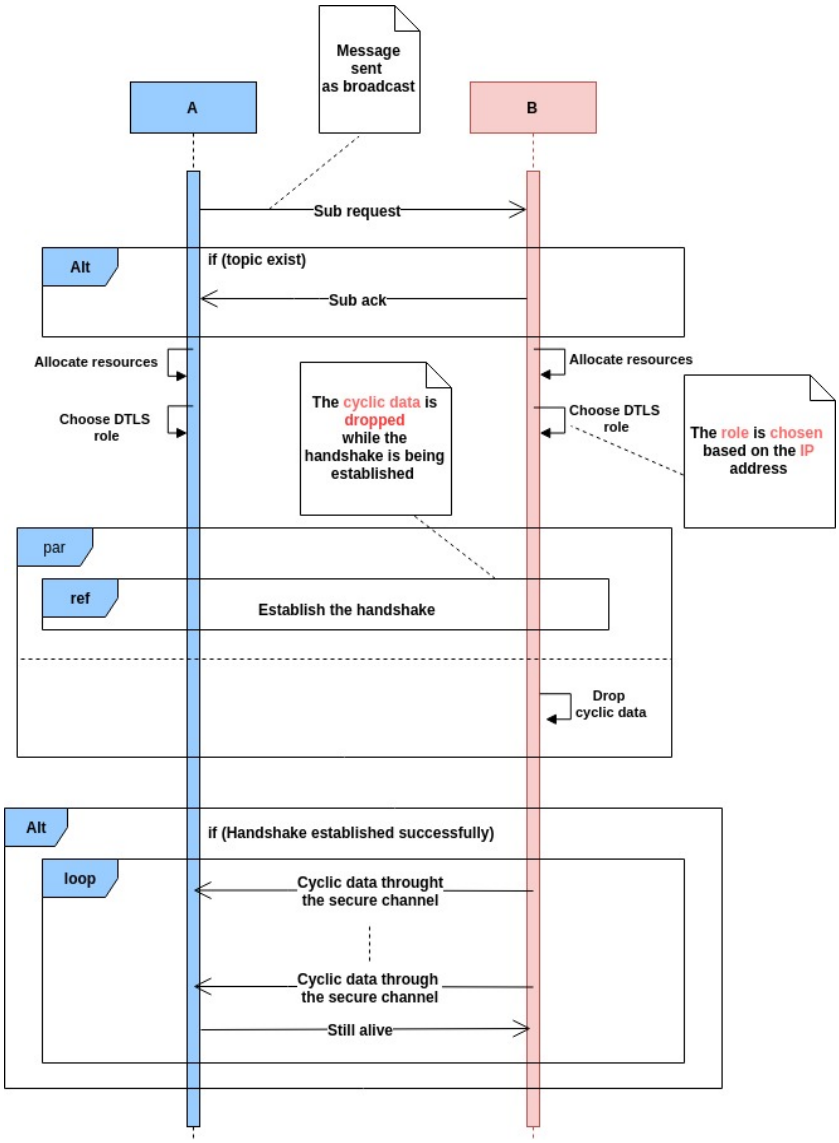

Fig. 4.3: *Nominal case of a secure communication*

The Figure 4.3 illustrates a nominal case of a secure communication between two SafetyNETp devices. Once the subscription process is done, both of the devices allocate the necessary memory needed for the secure channel and then they choose DTLS roles based on their IP addresses. The handshake then starts, and the *cyclic data* is dropped. Once the handshake is finished successfully, sending the *cyclic data* is resumed to be sent through the secure channel.

4.2 Detailed Design

4.2.1 Security Layer Positioning

In this section, we see the positioning of the security layer and SafetyNETp in the TCP/IP model and we discuss according to the messages' types whether a message should be sent through the security layer or not.

As illustrated in Figure 4.4, the security layer is between UDP (transport layer) and SafetyNETp (application layer). Sending a SafetyNETp application message can follow two paths, whether to be packed directly into a UDP datagram or packed into our security layer frame and then into UDP. One can ask why would we enable SafetyNETp to skip the security layer and use directly UDP. As we saw before in order to switch to a secure communication, the DTLS handshake need to be established. The trigger of the handshake establishment is the first subscription process that's why the first subscription messages are sent unprotected because at this level no secure channel exists.

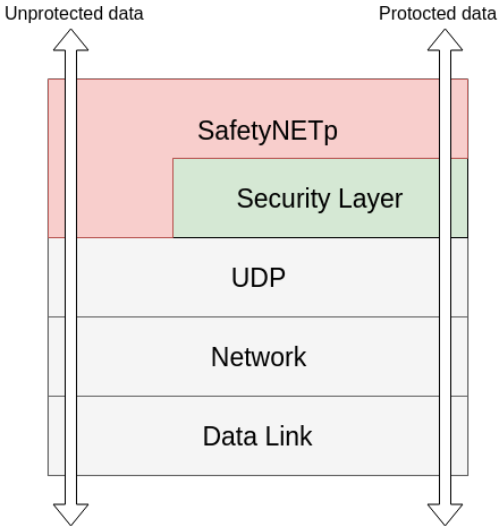

Fig. 4.4: *The security layer and SafetyNETp in TCP/IP model*

Once The handshake is established, a secure channel is provided and data could be exchanged securely but what are the messages that should be sent through the secure channel? Do all of them should be sent securely? Regarding *subscribe requests* and *acknowledgments* they obviously need to be sent securely, otherwise, a fake *subscribe request* could be sent and unnecessary data will be exchanged which leads to more CPU processing in both sides (publisher and subscriber) and more bandwidth usage. Actually, an attacker could use the fact that the subscription messages are unprotected. Being unprotected means that no integrity checks are performed, therefore, if

an attacker modifies the content of a message, the modification will not be detected. A possible attack is shown in Figure 4.5, in this example, the attacker could modify the content of the packets with a man in the middle attack. When the subscriber sends a *subscribe request* for a topic X, the attacker intercepts the messages and modifies it by changing the topic to Y. When the publisher receives the message it sends back an acknowledgment and starts sending *cyclic data* for topic Y which is not needed by the subscriber. Moreover, an attacker could entirely forge and send fake *subscribe requests* and, therefore, the bandwidth could be saturated. The *subscribe requests* are sent as broadcast to all the devices present on the network as the topic owner is unknown in advance. Each secure channel corresponds to a couple of a publisher and a subscriber and there is no secure channel shared between all the devices. Therefore, *subscribe request* messages could not be sent securely as broadcast. In order to avoid the previous problem, alongside sending the *subscribe requests* as broadcast, the subscriber also sends the same *subscribe requests* as unicast and secured to the publishers with which it has an available secure channel. When a publisher receives a *subscribe request* as broadcast and unprotected from a subscriber, if it has an available secure channel, the publisher ignore the subscribe request. If the publisher has no channel, it goes through the handshake establishment. Hence, the publisher could avoid sending *cyclic data* when receiving a tampered or a forged subscribe request.

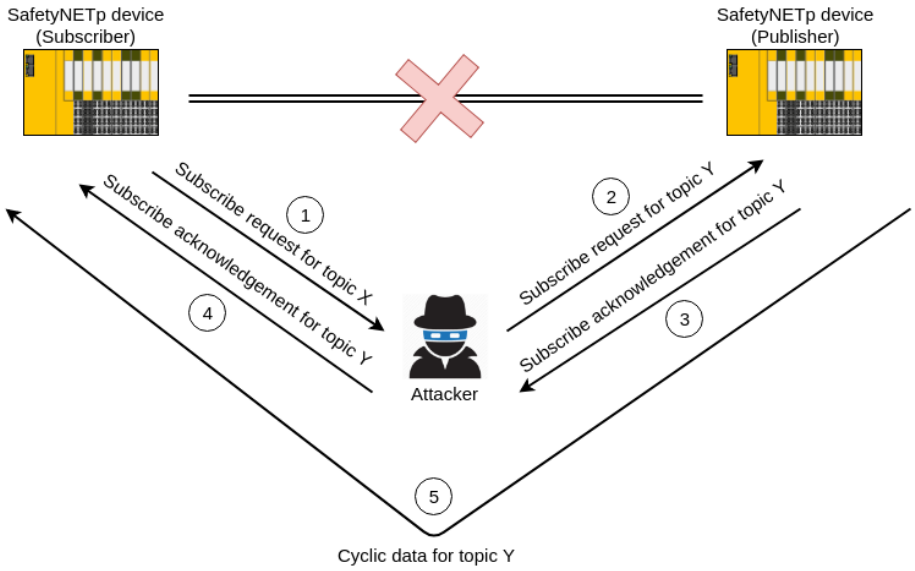

Fig. 4.5: *Man in the middle attack exploiting unprotected subscription*

As the rest of the messages are sent as unicast, they could be secured easily unlike the *subscribe request* message. The *unsubscribe* and *unpublished* messages could cause issues if they are unprotected. In fact, an attacker could send a fake *unsubscribe request* to a publisher which will stop sending *cyclic data* upon receiving the message. The last message type is the *still alive* message, as we discussed

before it is used by the subscriber to inform the publisher that it is still alive. Therefore, if it is sent unprotected, when a subscriber reboots for example, an attacker can send *still alive* messages to the list of publishers communicating with the subscriber. Hence they will not detect that the subscriber is down. To conclude after analyzing the situation for each message type, once the handshake is established all the messages should be sent through the secure channel.

4.2.2 Flowcharts for Outgoing Messages

In any SafetyNETp device using our security layer, whether the device is a publisher or a subscriber, each outgoing message should follow the flowchart in Figure 4.6.

Let's take the example of two SafetyNETp devices, a subscriber and a publisher respectively A and B. Initially, before starting the communication, no secure channel is available. The communication starts when A sends a *subscribe request* to B. The *subscribe requests* are sent unprotected as broadcast following the Path 4 and as no secure channel exists, no *subscribe request* will be sent protected at this level.

Upon receiving the *subscribe request*, like A, B will verify first if there is a secure channel for A. Obviously, no secure channel exists, therefore, the *subscribe acknowledgment* is sent unprotected and Path 1 is followed. After sending the *subscribe acknowledgment*, B will allocate the necessary resources and launch the handshake. If the handshake is launched as a server, B will just wait for Client Hello messages coming from A, if it is launched as a client, B will start sending Client Hello messages to A. Following the *subscribe acknowledgment*, B will start sending *cyclic data*. In this level, the handshake is being established, which means the identity of A is not verified yet. The *cyclic data* should be dropped, hence, Path 2 is followed. While establishing the handshake, each *cyclic data* message will follow Path 2 until it is finished. Once the first handshake is established (we will have more than one handshake, more details are given this in the next sections), every message is switched to be sent through the secure channel. All the messages except *subscribe requests* are sent following path 5. After that the handshake is established, following the path 4, the *subscribe requests* are sent unprotected as broadcast and secured as unicast through all the available channels. In this case, If A performs another subscription after that the handshake is established, B will receive two subscribe requests for the same topic, one as unicast through the security layer, and the other original unprotected broadcast subscribe request. Obviously B, ignore the one sent as broadcast and process only the secured request. As we mentioned before, sending *subscribe requests* secured as unicast, enables the publisher to avoid responding for tampered and forged subscribe requests.

The path 3 shows how messages other than *subscribe requests* and *cyclic data* are sent before and while establishing the handshake.

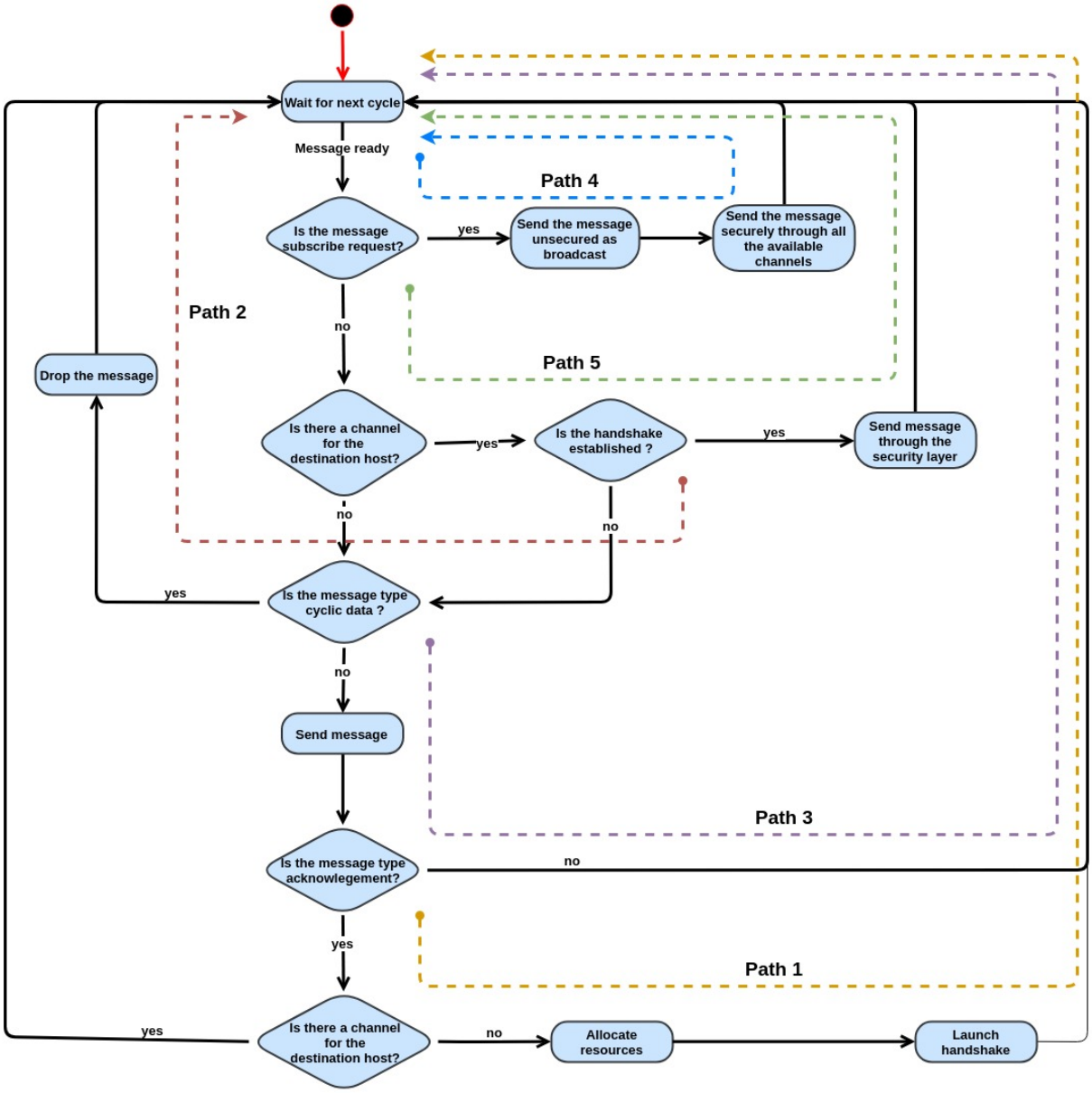

Fig. 4.6: *Flowchart for outgoing messages*

4.2.3 Flowchart for Incoming Messages

Following the flowchart in Figure 4.6 we ensure a secure encrypted and authenticated communication in which messages are sent only to authorized devices, However, this flowchart does not include the flow for received messages. The flowchart in Figure 4.7 illustrates how each incoming message is treated.

When *cyclic data* is received and no secure channel is available, the message is dropped as shown in Path 1. If we assume that we have a subscriber and a publisher respectively A and B, after sending a *subscribe request*, A will receive the first *subscribe acknowledgment*. Upon receiving it, as no channel exists, A will allocate the necessary resources and launch the handshake (Path 2). Meanwhile, if a *cyclic data* message is received it will be dropped as the identity of B is not verified yet (Path 1).

Once the first handshake is established, A and B will switch to communicate via the secure channel. When a message is received the identity of the sender as well as the message integrity will be verified. If the message is verified correctly, it will be decrypted and passed to SafetyNETp application layer to get processed (Path 3). If the message is not verified it will be dropped (Path 4).

When an attacker impersonates the identity of B for example and sends a *cyclic data* message to A, if no secure channel is available, the message will be dropped (Path 1). After the handshake is established, if the attacker finds a way to modify messages, the integrity checks will fail and the message will be dropped (Path 4).

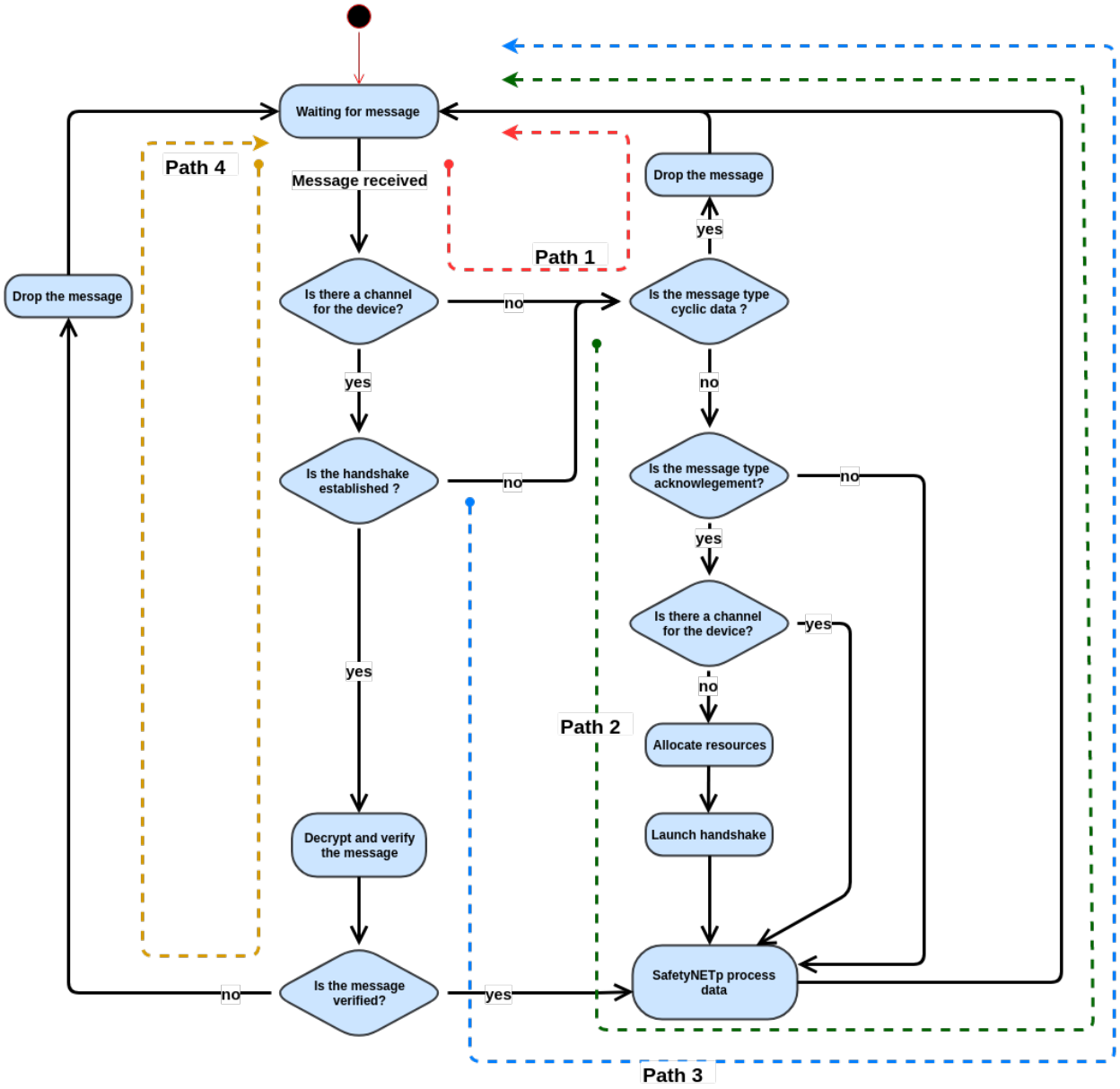

Fig. 4.7: *Flowchart for incoming messages*

4.2.4 First Subscription Vulnerability

Triggered by the first handshake process, the devices establish the DTLS handshake and start exchanging data securely. The trigger for establishing the handshake could not be secured as no secure channel exists before establishing the handshake. Therefore we are obliged to perform the first subscription unprotected as it is the trigger. We already mentioned the issues of sending unprotected subscribe requests. Fake *subscribe requests* could be sent which results on exchanging unnecessary data in the network. We already proposed a solution for securing subscribe requests after the secure channel is provided, however, we did not discuss the first subscription issue.

When the system starts up, before any message is exchanged, an attacker could easily impersonate the identity of a device and send a *subscribe request* with its identity. The device which has the topic will send back a subscribe acknowledgment as response and both devices will establish the handshake. As the first subscription is forged by the attacker, the *cyclic data* sent by the publisher is not needed by the subscriber.

In order to avoid receiving the unnecessary cyclic data, we propose two solutions. After that the handshake is established, we reset the application state of the two parts the publisher and the subscriber. By resetting the application state, both the publisher and the subscriber forget about the previous subscriptions and they perform the subscription process another time. Therefore, if the first subscription is fake, by resetting the application state, no *cyclic data* for the fake subscription will be sent and all the new subscription could be performed securely using the available channel. In fact, this solution could not be applied currently, as SafetyNETp does not provide an API for resetting the application. However, The solution is interesting and we could ask SafetyNETp developers for such API to solve the problem.

The second solution consists of changing the handshake timing previously discussed and perform the handshake at the second position shown in Figure 4.1. In this case, when a *subscribe request* is received, our security layer don't pass the message to SafetyNETp to be processed. Because if the topic exists the device will send a *subscribe acknowledgment* and start sending *cyclic data* which could be unnecessary as the subscription is not protected. Not passing the *subscribe request* to SafeyNETp allows avoiding this issue. The device which sends a *subscribe request* activates a server and the receiving device after verifying that the topic exists, takes the client role and start establishing the handshake with the subscriber. Using this method no subscribe acknowledgment is sent before establishing the handshake, therefore, no *cyclic data* is sent. After finishing the handshake, our security layer let SafetyNETp process *subscribe requests* and all the message could be exchanged securely. The problem with this method is that our security layer is not able to know whether a topic exists or not without letting SafetyNETp process subscribe requests. In fact, for our security

layer, the indicator for the topic existence is the subscribe acknowledgment. It is not possible to detect whether a topic exists or not without seeing a subscribe acknowledgment. Therefore to make this possible, SafetyNETp developers need to provide a function which returns whether a topic exists having a subscribe request as input.

To conclude, the first subscription process is currently performed without protection and the problem could be solved using one of the cited solutions if SafetyNETp developers could provide the needed APIs.

4.2.5 Keying Materials Renewal

Cryptology is a mathematical science which contains two branches, cryptography, and cryptanalysis. The main goal of cryptography is to ensure data confidentiality by encrypting an input message called plaintext to obtain an encrypted output called ciphertext. The obtained ciphertext is unreadable and it can be decrypted only by parts that have the secret key used for encryption. However, the cryptanalysis goal is to gain as much information as possible about the original unencrypted data. Cryptanalysis can lead to decrypt ciphertexts without having the secret key and much worse it can even lead to figure it out. This achieved by several technics, for example, analyzing a huge number of ciphertexts. This technic is called cryptanalysis with ciphertext-only, which mean that the attacker knows only ciphertexts [15].

Our security layer provides authentication, confidentiality, and integrity. These three services rely on keeping the keying material being used secret. In SafetyNETp networks, using our security layer, publishers send each cycle encrypted *cyclic data* messages to its subscribers. An untrusted third party can easily obtain a huge number of encrypted messages which allows him to perform cryptanalysis studies in order to break the encryption. If the keying materials are compromised, encrypted messages can be decrypted by an attacker, the messages can be modified without detecting the modification, an attacker will be able to impersonate the identity of a device legitimately.

The cryptanalysis relies on studying the encrypted traffic which needs a huge amount of encrypted message and another important factor which is time. To avoid this kind of attacks, we can simply renew the keying materials by running a new handshake every period of time. This period should be defined based on how much time an attacker needs to be able to break the encryption. Once the period is defined, each two communicating devices run a new handshake upon the expiry of the period. Figure 4.8 illustrates how keying material renewal is performed. After that the first handshake has been established, a timeout is set and *cyclic data* is sent using the first provided keying material or in other words the first DTLS context. Upon the timeout expiry, a new handshake is established in parallel while sending *cyclic data*. When the new handshake is established, we can

choose whether to switch directly to use the new DTLS context or to wait for a period of time before switching like shown the figure. At each handshake, the timer is reset and upon its expiry, a new handshake is established and therefore we can minimize the possibility of cryptanalysis attacks.

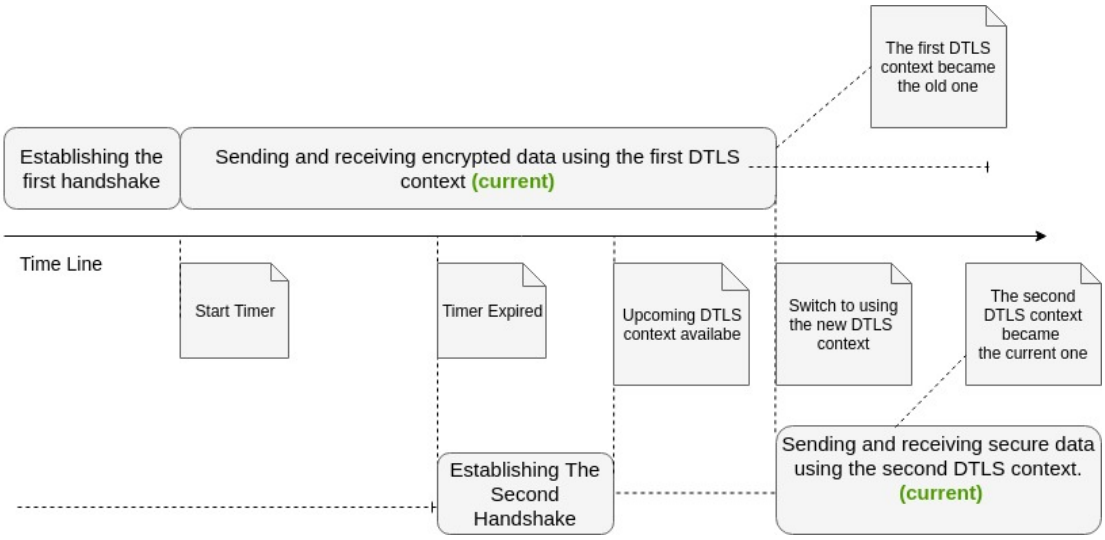

Fig. 4.8: *Keying Materials Renewal*

4.2.6 Session Closing

When no more data need to be exchanged between two devices, the session should be closed and the allocated resources for the handshake establishment and keying materials should be freed. A session could be closed for several reasons.

First, the subscriber could unsubscribe from all the topics received from a certain publisher, therefore, there will be no more data published to the subscriber. Hence the secure channel is not needed anymore. At this level, the session can be closed and the resources can be liberated. In each device, we need to keep track of the topics being used and once no topic is being communicated the session should be closed. In each Publisher/Subscriber relation, the topics are counted as follows. The subscriber increments the number of topics being used each time it receives a *subscribe acknowledgment* from the publisher. The subscriber decrements the number if it receives an *unpublished* message from the publisher or after having sent an *unsubscribe request*. The publisher applies the same principle, it increments the number after sending a *subscribe acknowledgment* and decrements the number if it receives an *unsubscribe request* or after sending an *unpublished* message. In every device when the number of topics being used for a session is equal to zero, the session will be closed. This works fine if we assume that messages are sent reliably, however, if we consider using unreliable transport we will have issues. Assuming we have a subscriber and a publisher and the

subscriber is subscribed for just one single topic, the number of topics being used on both sides is equal to one. If the subscriber sends an *unsubscribe request* to the publisher and the message is lost, the subscriber decrements the number of topics being used to become zero which is not the case for the publisher (the message is lost). In this case, we end up having different numbers on each side and only the subscriber closes the session. Alongside counting the topics, we add another condition to avoid this desynchronization. A session should be closed if two conditions are satisfied. For the subscriber, the number of topics being used should be equal to zero and in addition, no *cyclic data* is received. In fact, if the subscriber still receives *cyclic data* means that there has been a packet loss. Hence the subscriber can resend the *unsubscribe request* until the publisher stops sending *cyclic data*. For the publisher case, the session should be closed, like the subscriber, if the number of topics being used is equal to zero and if no *still alive* messages are being received.

The second case for which a session should be closed is that one of both sides is not working anymore (the devices crashes or reboots for example). After finishing subscription process, *still alive* messages are sent periodically by the subscriber to the publisher, hence, if the publisher does not receive *still alive* messages for a well-defined timeout (defined by SafetyNETp application layer), it considers that the subscriber is down. When the *still alive* message timeout expires, the session should be closed. Regarding the subscriber, it considers that the publisher is down if it does not receive *cyclic data* messages for also a well-defined timeout (defined by SafetyNETp application layer). When the timeout expires, the subscriber should close the session.

The last case is when performing a handshake. Whether it is the first handshake or a handshake for session renewal, if the handshake fails, the session should be closed.

We can have the situation where a publisher and a subscriber have just closed their session and the subscriber decides to get subscribed another time. As the session was closed, they need to re-establish the handshake in order to communicate. To avoid this situation, if no communication is needed between the publisher and the subscriber, the session will not be closed immediately. Instead, a timeout will be set and the session will be closed when the timeout expires. This allows more flexibility. While the timeout is not expired, the session will still open and could be used for a new communication. Hence, no need for a new handshake.

4.2.7 Frame Layout

After that the handshake is established SafetyNETp messages are no more sent as plaintext, instead, they are sent encrypted and encapsulated into DTLS frames. In order to minimize the possibility of cryptanalysis attacks, we decided to run a new handshake each a well defined period to renew the keying materials being used. Due to reordering, it is still possible to receive DTLS frames

encrypted by the old keying material from the previous handshake. Therefore, we are not able to know whether a DTLS frame is encrypted using the new or the old keying material. Hence the receiving device will not be able to decrypt and verify messages.

In order to solve this issue, our solution is to add additional information to the message to indicate which keying material was used for encryption. In fact, this consists of adding a one-byte field before the DTLS frame. This field is incremented each time the handshake is established. We call this field channel id and according to its value, the receiving device could decide which keying material to use to decrypt and verify the message.

Figure 4.9 shows the security layer frame layout. The first byte indicates that the message is a secured frame, the second byte is channel id which indicates which keying material was used for encryption. The rest of the message is the encrypted DTLS frame which encapsulates SafetyNETp application layer message.

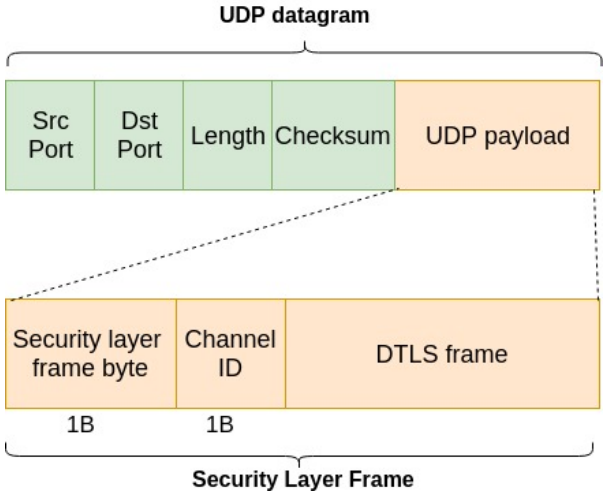

Fig. 4.9: *The security layer frame layout*

Conclusion

Throughout this chapter, we presented the global and the detailed design where we introduced the existing problems, we provided the candidate alternatives and chose the solutions we judge suitable for our case.

5 Realization

After designing our security layer, we move in this chapter to see the realization. We begin by presenting the software and the hardware environments, we move then to present how we implemented our design, and finally go through testing the implementation.

5.1 Work Environment

For realizing our project, we used a specific hardware environment as well as a software environment detailed in this section.

5.1.1 Software Environment

In order to implement and integrate our security layer, we used the following tools and software:

- **SafetyNETp library**: its code is written in C as it provides high performance and low-level memory management. Hence, we used the C programming language to implement our design.
- **GCC**: GNU Compiler Collection is a C compiler provided by GNU which we used to compile our C code. More specifically we used arm-linux-gnueabi-gcc compiler version 5.4.0.
- **make**: make is a GNU utility which allows facilitating the compilation process of a project using Makefiles.
- **Gdb**: Gdb is a debugger provided by GNU which allows to debug programs of several programming languages. We used Gdb to debug our C code.
- **Filezilla**: Filezilla is a graphical user interface for an FTP client, used to transfer the executable files from our machine to SafetyNETp devices.
- **SSH**: Used to connect remotely to SafetyNETp devices.
- **OpenSSL CLI**: Used to generate private and public keys, to create certificate authorities and to perform digital signatures.
- **Shell**: Used for commanding SafetyNETp devices and for automating the keying material and the certificates generation.
- **Git**: Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. We primarily used it for source code manage-

ment during development committing the several version of the code when we make our improvements.

- **Wireshark**: Wireshark is a free and an open-source packet analyzer. We used it to analyze the SafetyNETp and the security layer traffic.
- **lua**: Programming language which we used for developing a plugin for Wireshark to support SafetyNETp.
- **Vi**: A command line text editor used to edit SafetyNETp devices' files as no graphical user interface is available.
- **Atom**: Atom is an open source and free text and source code editor which provides a lot of features that facilitate code writing (highlighting code, auto-completion, include Git GUI ...). All of our source code is written used Atom.
- **Doxygen**: Doxygen is tool which allows to generate code documentation using comments in HTML or PDF format.

5.1.2 Hardware Environment

For the realization of this project, Pilz, the company which created SafetyNETp protocol, provided us with two Revolution Pis containing their SafetyNETp implementation. The Revolution Pi (Figure 5.1) is a small computer designed by the Kunbus german company based on the Rasberry Pi Compute Module. Unlike Rasberry Pi which is not designed for an industrial usage, the Revolution Pi is designed to fit in industrial environments. As shown in Table 5.1, the Revolution Pi has a protection against electrostatic discharge and electromagnetic interference according to the norms EN 61131-2 and IEC 61000-6-2. In operating mode, the Revolution Pi could support a temperature between -40 °C and +55 °C which exceeds the requirements of the norm EN 61131-2. The Revolution could be alimented by a power supply between 10.7 V and 28.8 V. These characteristics make the Revolution Pi suitable for industrial usage [16].

Processor	BCM2835, 700 MHz, single-core
RAM	500 MByte
Electrostatic discharge protection	4 kV / 8 kV (according to EN 61131-2 and IEC 61000-6-2)
Electromagnetic interference protection	Passed (according to EN 61131-2 and IEC 61000-6-2)
Surge/Burst tests	Passed (according to EN 61131-2 and IEC 61000-6-2)
Power supply	min. 10.7 V - max. 28.8 V
Operating temperature	-40 °C to +55 °C(exceeds EN 61131-2 requirements)
Storage temperature	-40 °C to +85 °C (exceeds EN 61131-2 requirements)

Tab. 5.1: *Revolution Pi characteristics [16]*

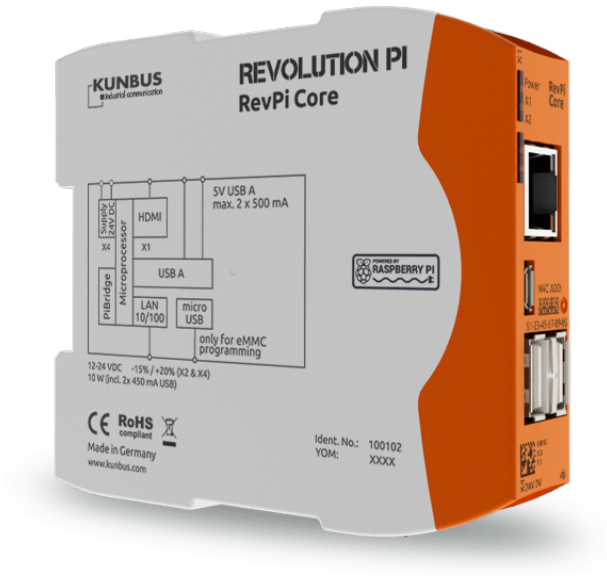

Fig. 5.1: *Revolution Pi*

For the development, we used a laptop with the characteristics shown in Table 5.2.

Processor	Intel core i5
RAM	8 Gbyte
Graphics	Intel HD Graphics 620

Tab. 5.2: *Used computer characteristics*

For interconnecting the Revolution Pis and our laptop we used a hub and Ethernet cables.

5.2 Implementation

In this section, we see the several steps we went through in order to implement our design.

5.2.1 DTLS Implementation Choice

Various DTLS implementations exists which support DTLS up to its final version, among these, we can cite:

- **OpenSSL** [17]: This is an open source project that provides a robust, commercial-grade and full-featured toolkit for SSL/TLS protocols. It is also a general-purpose cryptography library. The OpenSSL toolkit is licensed under an Apache-style license, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions.
- **WolfSSL** [18]: WolfSSL is a lightweight portable embedded SSL/TLS library written in ANSI C. With its small size, speed, and feature set, WolfSSL targets embedded systems and resource-constrained environments. It implements TLS and DTLS up to their latest versions which are currently TLS 1.2 and DTLS 1.2 and provides progressive ciphers such as ChaCha20, Curve25519, NTRU, and Blake2b. This software falls into the General Public License (GPL) and in case of commercial software products, a commercial license must be bought.
- **MatrixSSL** [19]: MatrixSSL is an embedded SSL and TLS implementation designed for small footprint IoT devices requiring low overhead per connection. The library is less than 50Kb on disk with cipher suites. It includes client and server support through TLS 1.2, mutual authentication, and implementations of RSA, AES, SHA, SHA-256 and more.
- **MbedTLS** [20]: formerly known as PolarSSL, it is an open source embedded SSL/TLS library provided by arm and written in C. Its small memory footprint makes it suitable for embedded environments. MbedTLS provides an up to date implementations of both TLS and DTLS. MbedTLS is primarily licensed under the Apache 2.0 license and a packaged version is available under the GPL as well. Therefore MbedTLS could be used for free for both commercial and non-commercial use.

In fact, we decide to use MbedTLS for the following reasons [20]:

- **Tiny and portable library**: As MbedTLS is written in C, it is portable and suitable for embedded environments.
- **Easy to understand and use with a clean API**: MbedTLS offers a readable code and a logical, clean, and well-documentated API. MbedTLS provides a reference and a high-level design for its API (using UML diagrams) making understanding and using the library a lot easier.
- **Easy to reduce and expand the code**: MbedTLS has no global code and has minimal coupling between its modules. So it's easy to just grab a part and drop it into an existing project or add new functionality.
- **Apache and GPL licenses**: MbedTLS could be used for free in open source and non-open source project as well as in commercial and non-commercial projects.

5.2.2 SafetyNETp Plugin Implementation for Wireshark

To be able to understand the behavior of SafetyNETp devices, the first step was to analyze the traffic exchanged between the two devices provided by Pilz. For this purpose, we used the well-known traffic analyzer Wireshark. As SafetyNETp is proprietary protocol, it is difficult to find information about its functioning or its packet format in public sources. In fact, Wireshark does not recognize SafetyNETp packets, hence, the messages are shown as raw data. Analyzing raw data messages makes our task extremely difficult. Figure 5.2 shows a Wireshark capture of a data exchange between two SafetyNETp devices. As the figure shows, no message type is mentioned, Wireshark just presents SafetyNETp's data as UDP payload.

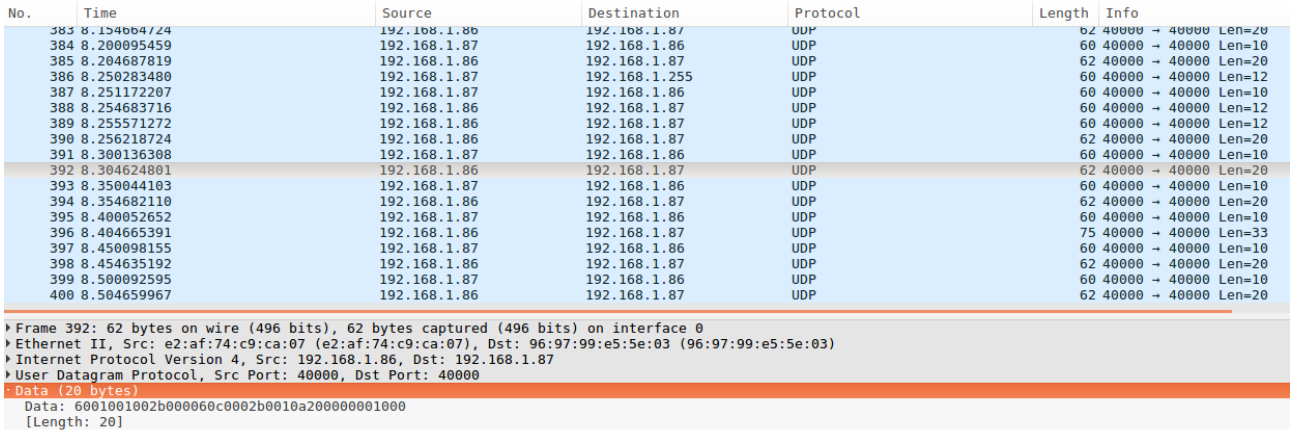

Fig. 5.2: *Wireshark capture without our plugin*

To facilitate analyzing and debugging the exchanged messages, we implemented a Wireshark plugin using lua programming language which enables the support of SafetyNETp in Wireshark. The Figure 5.3 shows a capture after we have integrated our plugin into Wireshark. The exchanged messages are no longer shown as raw UDP payload, instead, our plugin takes the responsibility for interpreting SafetyNETp packets and shows their types. Furthermore, we are able to use filters, allowing for example to see only specific messages types.

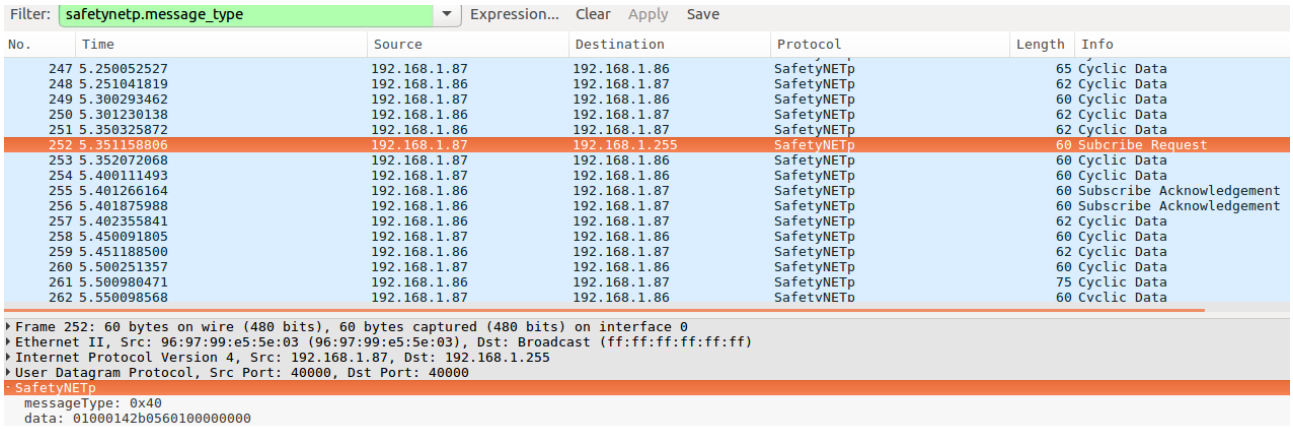

Fig. 5.3: *Wireshark capture after integrating our plugin*

5.2.3 Threads and Socket Managment

As the Revolution Pis run Linux Debian, for exchanging data over the network, SafetyNETp uses the Linux socket API. In fact, a socket is an abstraction through which an application may send and receive data, in much the same way as an open file allows an application to read and write data. A socket allows applications to communicate over the network, an information written to a socket by an application on one machine can be read by an application on a different machine and vice versa. Numerous types of sockets exist, for our case, we will be using UDP sockets.

Every SafetyNETp device using our security layer implementation runs several threads. As shown in Figure 5.4, four kinds of threads are present:

- **The first handshake thread**: This could be a client or a server thread, its role is to establish the first handshake. This thread terminates when the handshake finishes.
- **The receiving data thread**: This thread is responsible for receiving, decrypting, and verifying data. The thread terminates when its session is closed.
- **Session renewal thread**: This could be a client or a server thread, its role is to establish a new handshake in order to renew the keying material. This thread terminates when the handshake finishes or its session is closed.
- **Sending data thread**: This thread is responsible for sending both protected and unprotected data. This thread runs while the device is on.

In each device, one single thread exists for sending data, and n threads exist for receiving data, with n the number of channels. In fact for each connection, a *first handshake thread* is created for establishing the first handshake, then a *receiving data thread* is created for receiving, decrypting,

and verifying data, furthermore, periodically, a *session renewal thread* is created to renew the keying material.

For sending and receiving data, we have two possibilities, whether we create one single socket for all devices, or create and dedicate a specific socket for each device. A socket has separate buffers for writing and reading which means that it is possible to have two threads, one writing and one reading at the same time. However, having two threads writing to a socket at the same time is not possible. We could have the case where a device is for example establishing multiple handshakes with n devices at the same. In this case, n handshake threads will be running at the same time. If we use one single socket, we need to manage the threads access for writing into the socket as it is not possible to perform more than one write operation at the same time. Furthermore, when receiving a packet, our application has to filter the incoming packets to know their sources as all the devices send their data on the same socket. That's why, as shown in Figure 5.4, a separate socket is created for each device. Hence, having multiple handshake threads does not cause access problem anymore as each thread will be writing on a separate socket. In addition, no filtering is needed, the linux kernel will filter the packets according to the sockets.

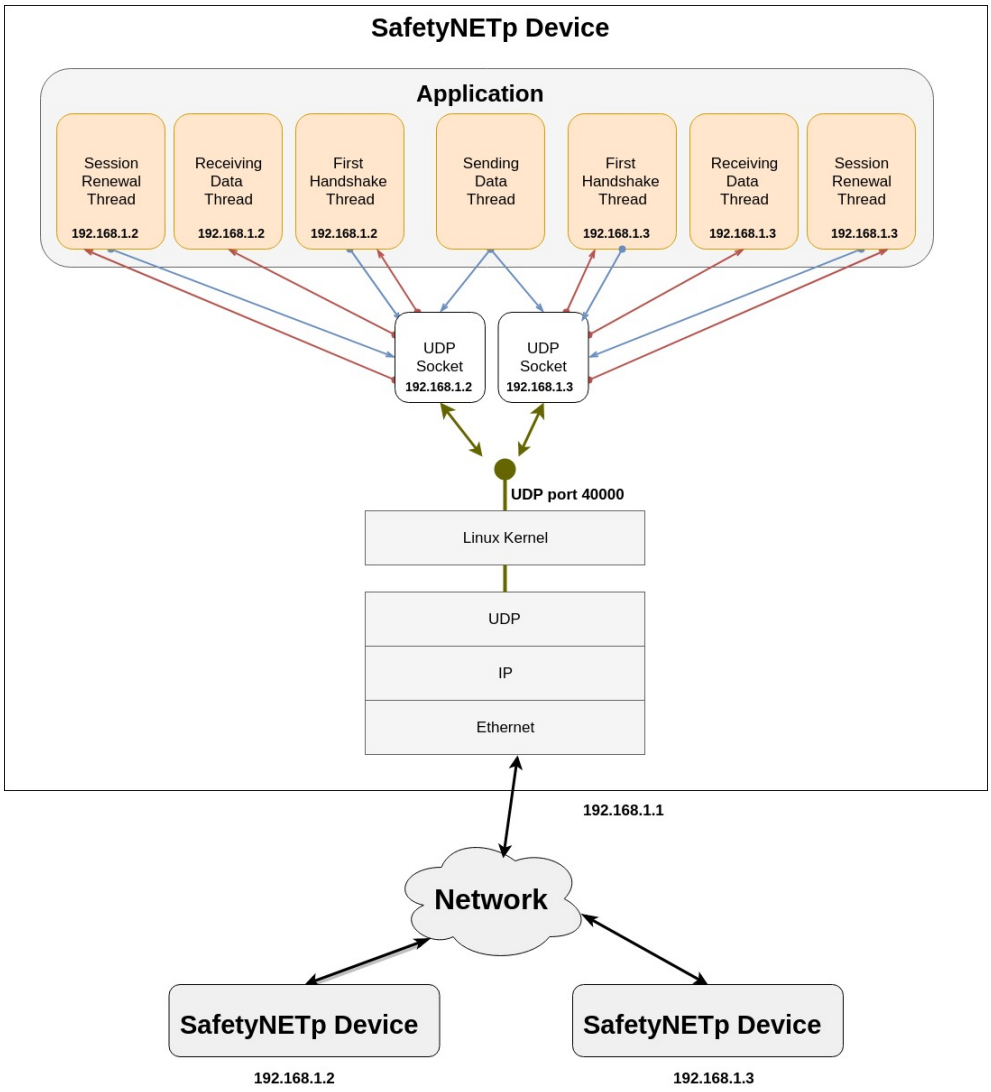

Fig. 5.4: *Threads and sockets*

As mentioned in the previous chapter, we need to renew periodically the keying material to minimize cryptanalysis attacks. This corresponds in our implementation to create a *session renewal thread*. It is obvious that the *session renewal thread* needs to write and read data from the dedicated socket. Meanwhile, the *sending data thread* is sending cyclically *cyclic data* which means that it is writing to the socket. Therefore, we could end up having simultaneous access to the socket. We can manage the simultaneous access by using a mutex variable to protect writing on the socket, in fact, this solves the problem by letting only one thread writing at a time. Figure 5.5 shows how the socket concurrent access is managed. If we assume that the green state is the current, in this case, the *sending data thread* will wait until the *session renewal thread* writes and unlocks the socket. Such management could delay sending *cyclic data* that's why we need to avoid this method.

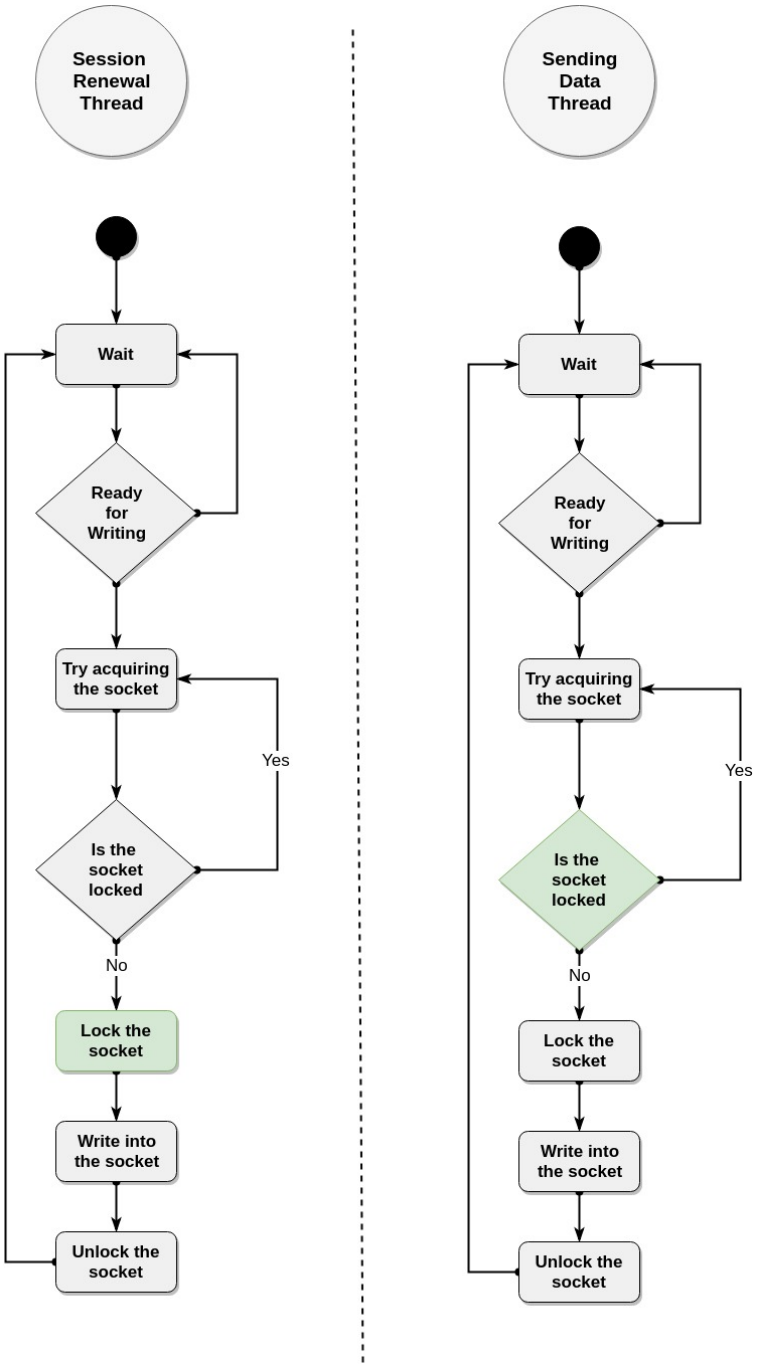

Fig. 5.5: *Socket concurrent access*

To avoid delaying *cyclic data*, we will not allow the *session renewal thread* to access the socket directly. Instead, it writes the message on a queue and only the *sending data thread* will have access to the sockets. When the *sending data thread* sends a message, it verifies the queue which contains the messages written by the *session renewal thread*. If the queue is not empty, the *sending data thread*

dequeues the first message and sends it. Using this method, we ensure that the *cyclic data* is sent immediately without any additional delay (Figure 5.6).

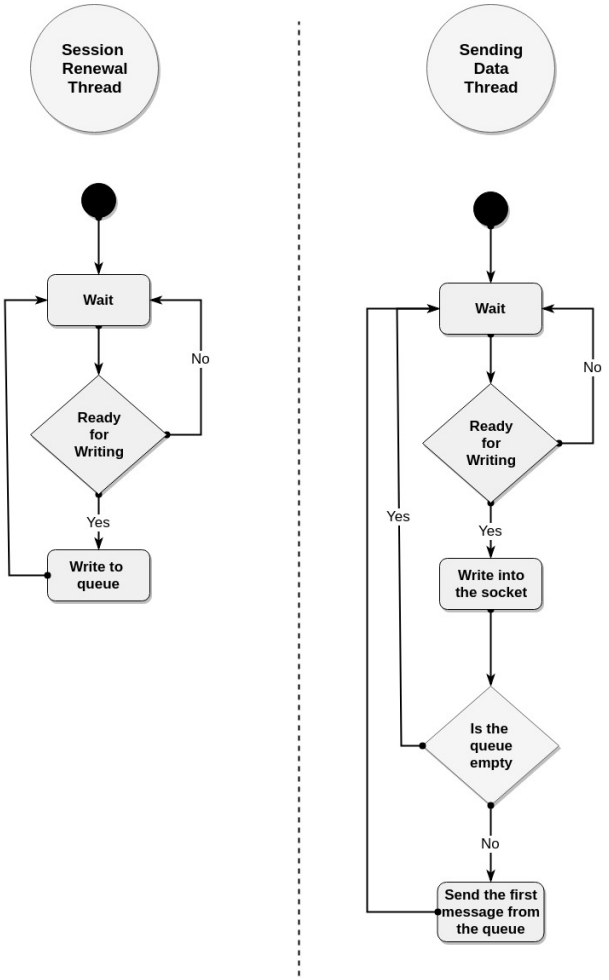

Fig. 5.6: *Prioritization of cyclic data messages over handshake messages*

5.2.4 Channel Lookup

Each SafetyNETp device can obviously communicate with multiple other devices. For instance, a publisher could be publishing data for an arbitrary number of n subscribers. The publisher will have a list of secure channels, each one is used to communicate securely with one subscriber. Each channel has as an identifier the IP address of the subscriber. For each *cyclic data* message to be sent, the publisher has to fetch the list of channels and to find the corresponding channel to use for the destination devices. Having a big number of channels could affect the cycle time that's why we will be interested in this section in selecting a search algorithm that allows keeping the cycle time as it is as much as possible.

In order to find the channel to use, we need to find the one which has the corresponding IP address which is no more than a 4 bytes integer. As a result, the problem is about finding an integer in a given list. The classic method for searching is linear search, given a list of n elements, it sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched. the complexity of the linear search method is equal to $O(n)$.

Another method is commonly used which is the binary search tree, it searches in a sorted list by repeatedly dividing the search interval in half. Begin with an interval covering the whole list. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty. The complexity of the binary search method is $O(log(n))$ if the tree is balanced, however, if it is unbalanced it could reach a complexity of $O(n)$.

To get rid of the balancing problem, AVL is the solution. AVL tree is a self-balancing binary search tree where the difference between heights of left and right subtrees cannot be more than one for all nodes. Most of the binary search tree operations (search, max, min, insert, delete, etc) take $O(h)$ time where h is the height of the binary search tree. The cost of these operations may become $O(n)$ for a skewed Binary tree where n is the number of nodes. If we make sure that height of the tree remains $O(log(n))$ after every insertion and deletion, then we can guarantee an upper bound of $O(log(n))$ for all these operations. The height of an AVL tree is always $O(log(n))$ where n is the number of nodes in the tree.

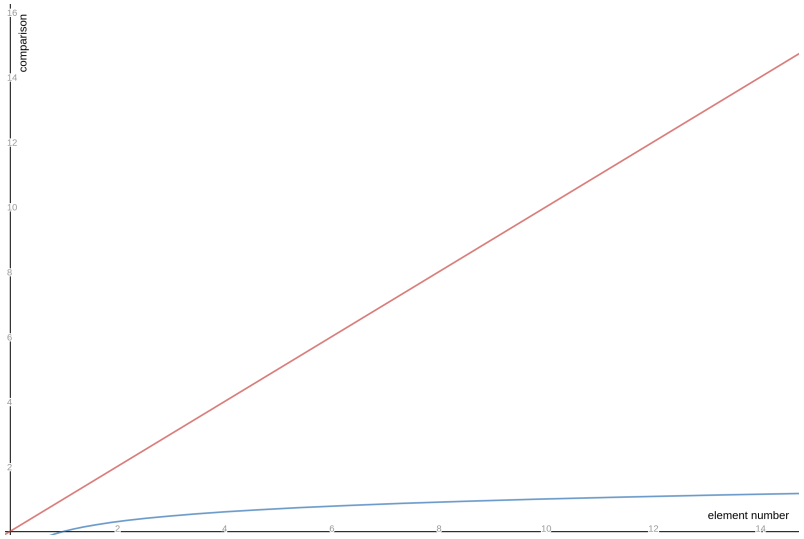

Fig. 5.7: *Comparison between $O(n)$ and $O(log(n))$*

Figure 5.7 shows the two functions n in red and $log(n)$ in blue it shows in other words how increasing the number of elements will affect the number of comparisons to be performed in

order to find the target. It is clear that $log(n)$ is always below n which means that the number of comparisons is always smaller. As a result, we used the binary search tree method.

5.3 Implementation Demo and Performance Test

As shown in Figure 5.8, our test environment consists of two Revolution Pis, a power supply and a hub for interconnecting the Revolution Pis with our laptop. The Revolution Pis are configured to subscribe mutually to each other, they have 192.168.1.86 and 192.168.1.87 as IP addresses. In this section we go into an overview to see the implementation demo, then we move to the test.

Fig. 5.8: *Test environment*

5.3.1 Demo

The Figure 5.9 shows the case where the communication is initiated and the first handshake is established. In fact, as we mentioned in the design chapter, the DTLS roles are chosen based on the IP addresses. The right capture is the screen of the device which has the IP address 192.168.1.87 and left one corresponds to the device which has the IP 192.168.1.86. As we can see the device with the higher IP launched a server thread whereas the device with the smaller IP launched a client thread. After launching the threads, they begin establishing the handshake and they exchange their certificates. Thereafter, each end-point verifies the other end-point's certificate. As we see in this capture, the certificates are verified correctly and therefore they start exchanging secured *cyclic data*.

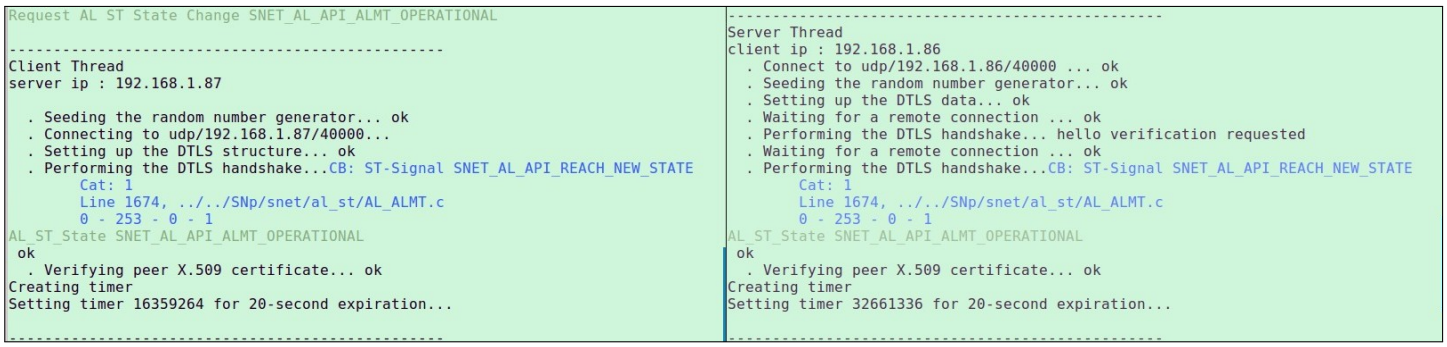

Fig. 5.9: *The first handshake establishment*

In order to minimize the risk of cryptanalysis attacks, as mentioned in the previous chapter, a new handshake is run each a well defined period of time. For test purposes, we set the period of session renewal to 20 seconds. As shown in Figure 5.10, each 20 seconds the devices launch a *renewal session thread* in order to establish a new handshake. Following each established handshake, the *channel id* header field is incremented to allow the receiving devices to verify and decrypt the messages with right keying material.

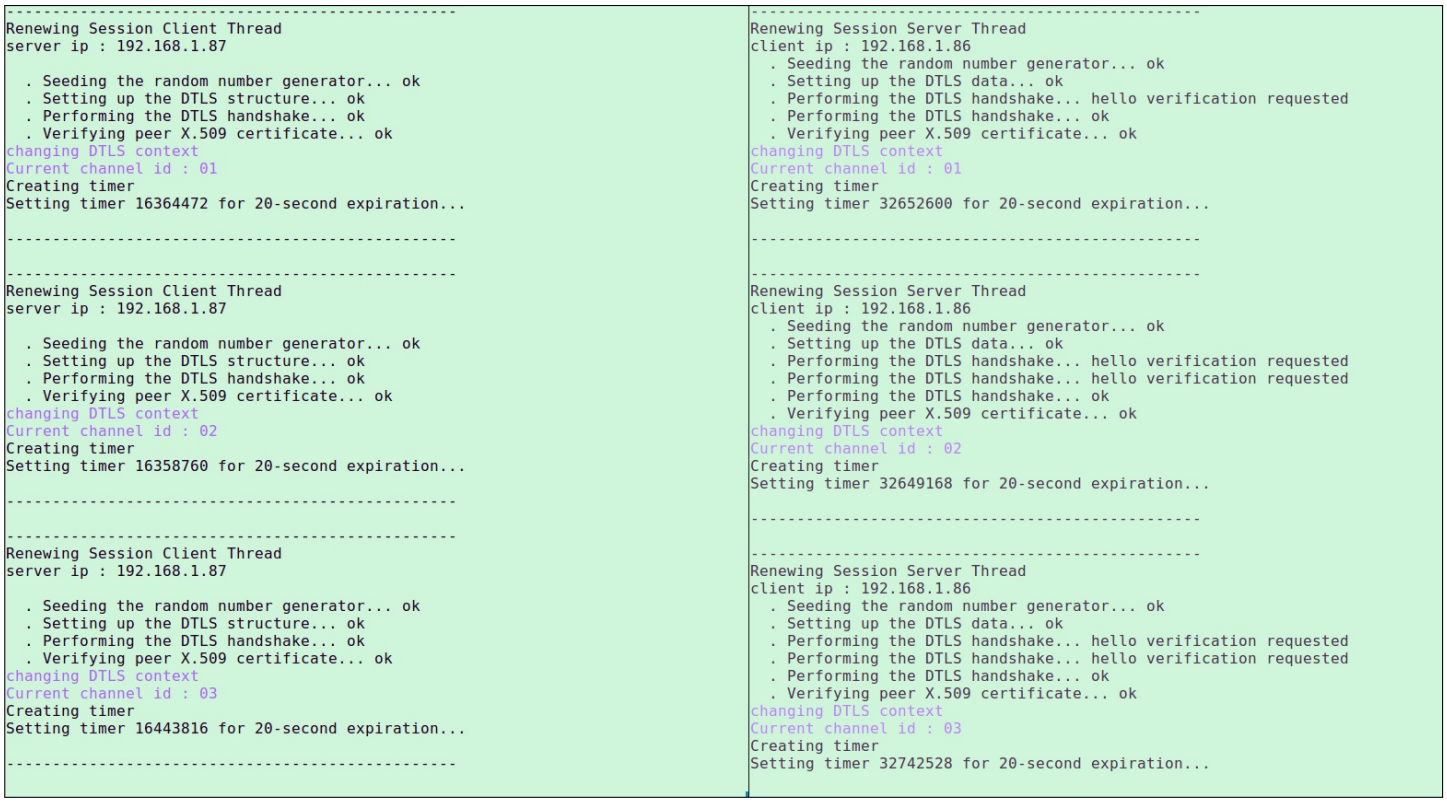

Fig. 5.10: *Session renewal threads*

The main difference between the *first handshake thread* and the *renewal session thread* is that while establishing the first handshake the cyclic data is blocked as the identity of the second end-point could not be verified before finishing the handshake. However, when renewing the keying material, the cyclic data is sent, and the new handshake's messages should be sent without interrupting sending cyclic data. The Figure 5.11 shows the message which SafetyNETp application layer logs when the timeout of the cyclic data expires. As shown in Figure 5.10 no such message is logged which confirms that the handshakes are being established without interrupting sending the cyclic data.

```
Verbindungskontolle: 2/3 Verbindungen vorhanden
CB: ST-Signal SNET_AL_API_PDO_TIMEOUT
        Cat: 3
        Line 104, ../../SNp/snet/al_st/AL_RxPDO_Heartbeat.c
        0 - 176128 - 1 - 0
Verbindungskontolle: 1/3 Verbindungen vorhanden
CB: ST-Signal SNET_AL_API_PDO_TIMEOUT
        Cat: 3
        Line 104, ../../SNp/snet/al_st/AL_RxPDO_Heartbeat.c
        0 - 176129 - 2 - 0
```

Fig. 5.11: *SafetyNETp timeout log*

The Figure 5.12 shows a Wireshark capture of a cyclic data message sent unprotected as plaintext before integrating our security layer. The Figure 5.13 shows a Wireshark capture of a cyclic data message sent through our security layer. The first byte of the secured message is the security layer frame byte which is, in this case, *ox88*, we chose arbitrarily this value and it could be changed via a configuration file. The second byte is the channel id and the rest of the message is the SafetyNETp cyclic data message encapsulated into a DTLS frame. We remark that the secured message is longer than the original message which is obvious as additional data are included for encryption and for integrity checks.

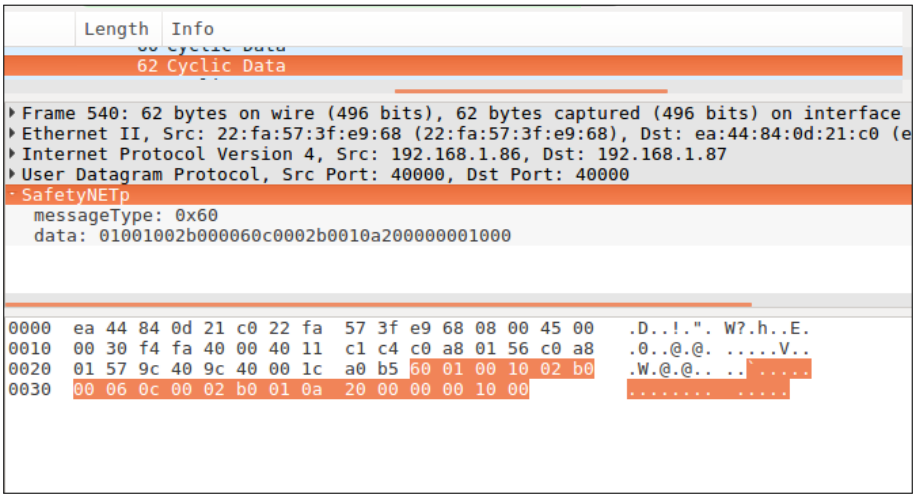

Fig. 5.12: *SafetyNETp original cyclic data message*

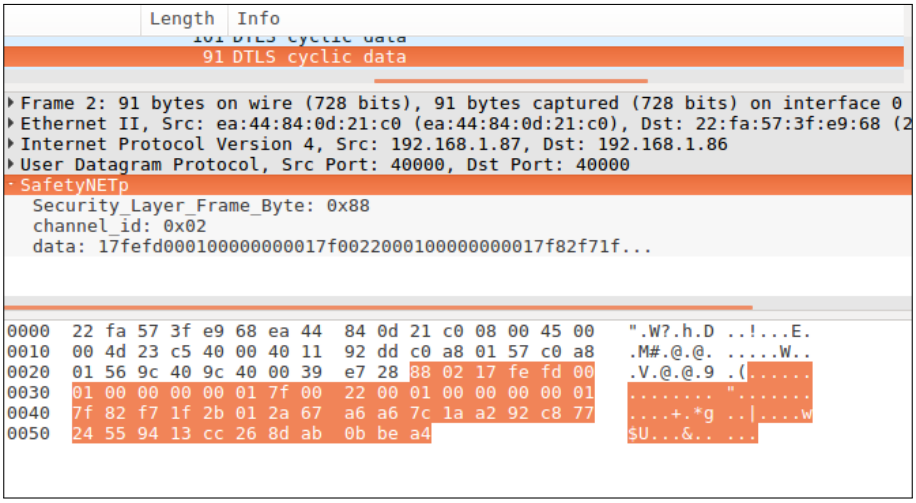

Fig. 5.13: *Secured cyclic data message*

In our security layer, when performing the DTLS handshake the devices' identities are verified using the certificate mechanism. Therefore we have created our own Public Key Infrastructure (PKI) in order to issue the devices' certificates. Our public key infrastructure is composed from two CAs, a root CA and an intermediate CA. The first step was to create the root CA by generating its public and private keys and generating its certificate. In fact, the root CA's certificate is self-signed as it is the first created CA. The second step was to create the intermediate CA, like creating the root CA, we generated the public and the private keys as well as the certificate. However, the intermediate CA's certificate is not self-signed, it is actually signed by the root CA. One could ask why do we need an intermediate CA when only the root CA is sufficient? In fact, it is a best practice to create an intermediate CA. The intermediate CA is used to issue and sign certificates on behalf of the root CA. This minimizes the risks of compromising the root CA's private key. After the creating the CAs, we issued a certificate for each device, thereafter, we configured the devices and added the root CA among the list of trusted CAs. Creating the CAs as well as issuing the certificates is performed using three bash scripts that we have developed. Figure 5.14 shows a certificate example which is, in this case, the intermediate CA's certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=DE, ST=Deutschland, O=Root
        Validity
            Not Before: Apr 25 13:28:33 2018 GMT
            Not After : Apr 22 13:28:33 2028 GMT
        Subject: C=DE, ST=Deutschland, O=Root, CN=safety
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:a4:da:17:ee:f6:43:77:12:e3:c5:cf:99:88:bd:
                    2b:5a:1c:93:95:98:2c:fd:0c:0b:8a:bc:73:d5:40:
                    0f:2c:3d:07:b4:db:68:1f:cb:ec:56:54:81:ea:77:
                    5e:f6:92:6d:95:59:82:c6:b1:8a:ec:5c:68:bf:0a:
                    f9:51:6f:78:82:6d:7b:c9:3d:14:a8:d3:1a:0f:3f:
                    12:87:ea:0e:b1:5a:3e:2d:c2:22:80:7c:df:9f:57:
                    bb:f1:d0:3b:4d:84:b7:6f:f9:bf:b9:93:4a:4b:aa:
                    ----------------------------------------
                    a9:af:26:56:4f:bf:7f:ea:75:1f:45:16:01:73:9b:
                    f5:90:af:96:c6:b6:b8:f8:cd:8a:cb:92:0c:63:44:
                    81:2c:8b:02:4c:65:0f:d7:a0:cc:a4:87:cd:ac:56:
                    dc:ee:6f:cf:ba:7a:de:f0:13:3e:ee:f9:6c:82:57:
                    af:05:05
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                58:0A:3B:7F:3E:08:77:1A:7F:A7:3C:22:64:A7:33:62:24:06:DB:68
            X509v3 Authority Key Identifier:
                keyid:41:85:3F:62:19:E8:39:2B:D8:79:23:0B:08:AE:C4:A6:05:27:6C:57

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         38:43:6d:cb:c3:66:bd:ec:d1:2b:5e:4e:88:fd:bf:ae:50:25:
         3c:8a:59:a1:ad:2b:71:18:94:ab:f7:04:34:e1:8a:7d:d5:64:
         18:27:29:2a:14:ff:c6:79:e1:50:40:44:f8:b9:61:f5:0e:c7:
         12:54:4b:31:8f:4e:d0:d4:f9:d1:87:2c:d0:35:ea:0f:bd:26:
         ----------------------------------------------------
         e8:22:46:de:31:87:83:65:8b:9e:d3:f6:73:c2:81:61:79:9a:
         59:4c:9b:35:ed:e6:8d:08:d4:ce:ee:bf:d3:71:ec:9c:0e:7a:
         e8:c0:a6:e3:05:09:15:55:92:a0:cf:a2:ef:ce:ec:78:cd:f8:
         b3:e1:59:c2:da:16:da:f8
```

Fig. 5.14: *Certificate example*

In order to facilitate modifying the parameters of the security layer, we created the configuration file shown in Listing 5.1. Through this configuration file, SafetyNETp developers could easily change the security layer frame byte, set the interval for renewing the keying material, set the max channel id value, print the sent and the received messages and print threads log to facilitate debugging.

```
/**
 * @brief Set Security Layer Frame byte value
 */
#define SECURITY_LAYER_FRAME_BYTE             0x88
/**
 * @brief Configure the renewing session interval in seconds
 */
#define RENEWING_SESSION_INTERVAL              20
/**
 * @brief Configure the period of time to wait after renewing the session to
 * switch using the new DTLS context
 */
#define DTLS_CONTEXT_SWITCH_TIME               4
/**
 * @brief Set the max value for the channel ID, when this value is reached
 * the next value will be 0x00
 */
#define MAX_CHANNEL_ID_VALUE                   0xff
/**
 * @brief Define this macro to print the Handshakes log
 */
#define PRINT_HANDSHAKE_LOG
/**
 * @brief Define this macro to print the Threads log
 * (Server and Client IP, channel ID, renewing session timer)
 */
#define PRINT_THREADS_LOG
/**
 * @brief Define this macro to print the outgoing messages (plaintext)
 */
#define PRINT_O_MSG
/**
 * @brief Define this macro to print the incoming messages (plaintext)
 */
#define PRINT_I_MSG
```

Listing 5.1: Security layer configuration file

5.3.2 Performance Test

In this section, we present the results of the performed tests which consist of evaluating SafetyNETp with and without the security layer. For this purpose, we have generated random messages with

different lengths and we have calculated the time needed for sending and receiving them. In order to have more accuracy, we have calculated the send and the receive time 1000 times for each length. The results presented in the following graphs show the average value for each message length.

As a first step, we started by evaluating SafetyNETp before integrating the security layer. Figure 5.15 shows the time needed for sending and receiving the messages.

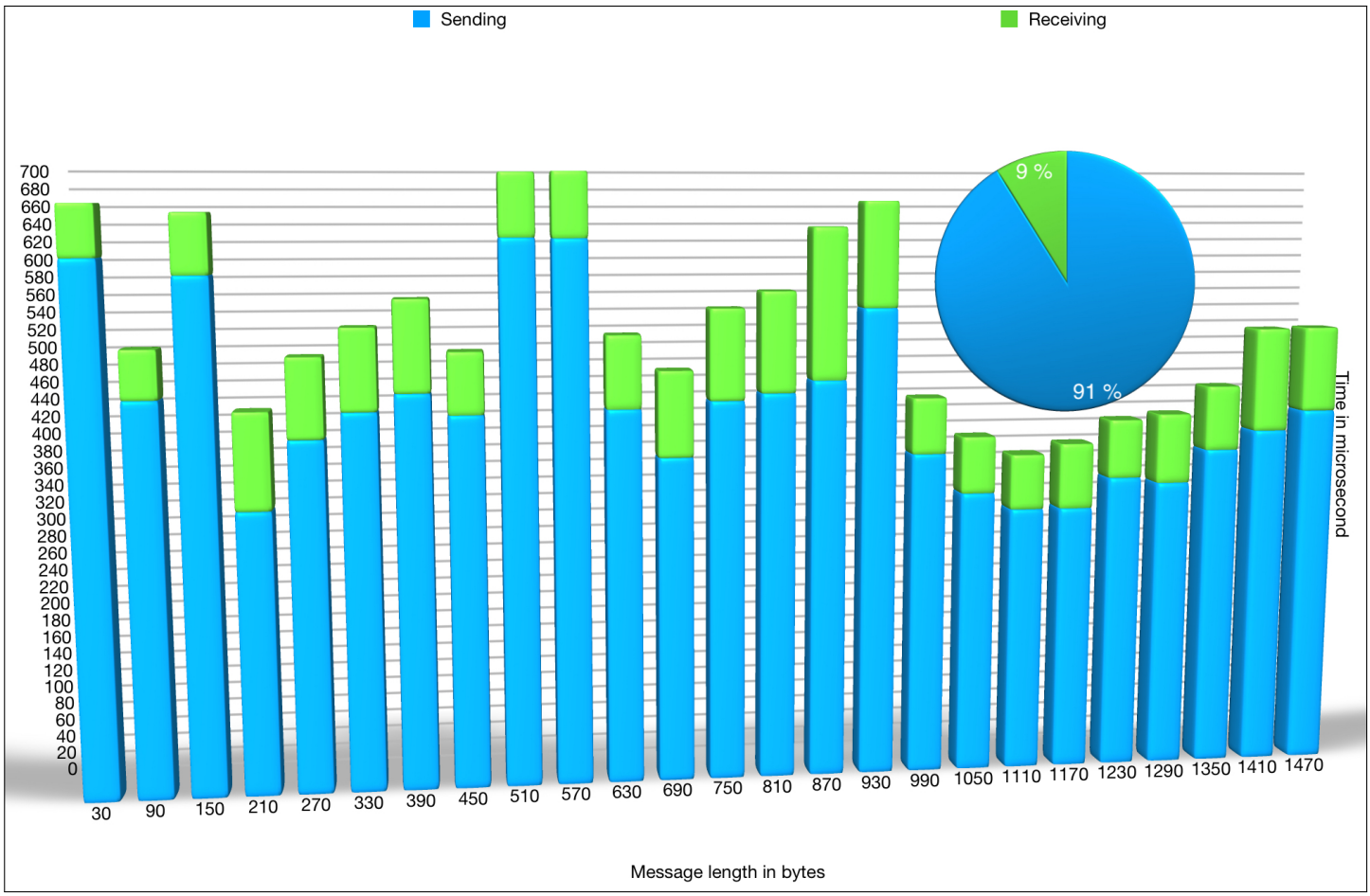

Fig. 5.15: *SafetyNETp before integrating the security layer*

As a second step, we evaluated our security layer. Figure 5.16 shows the time needed for sending and receiving messages through the security layer using TLS_RSA_WITH_AES_128_CBC_SHA256 cipher suite. The time includes the encryption, the MAC calculation and the decryption.

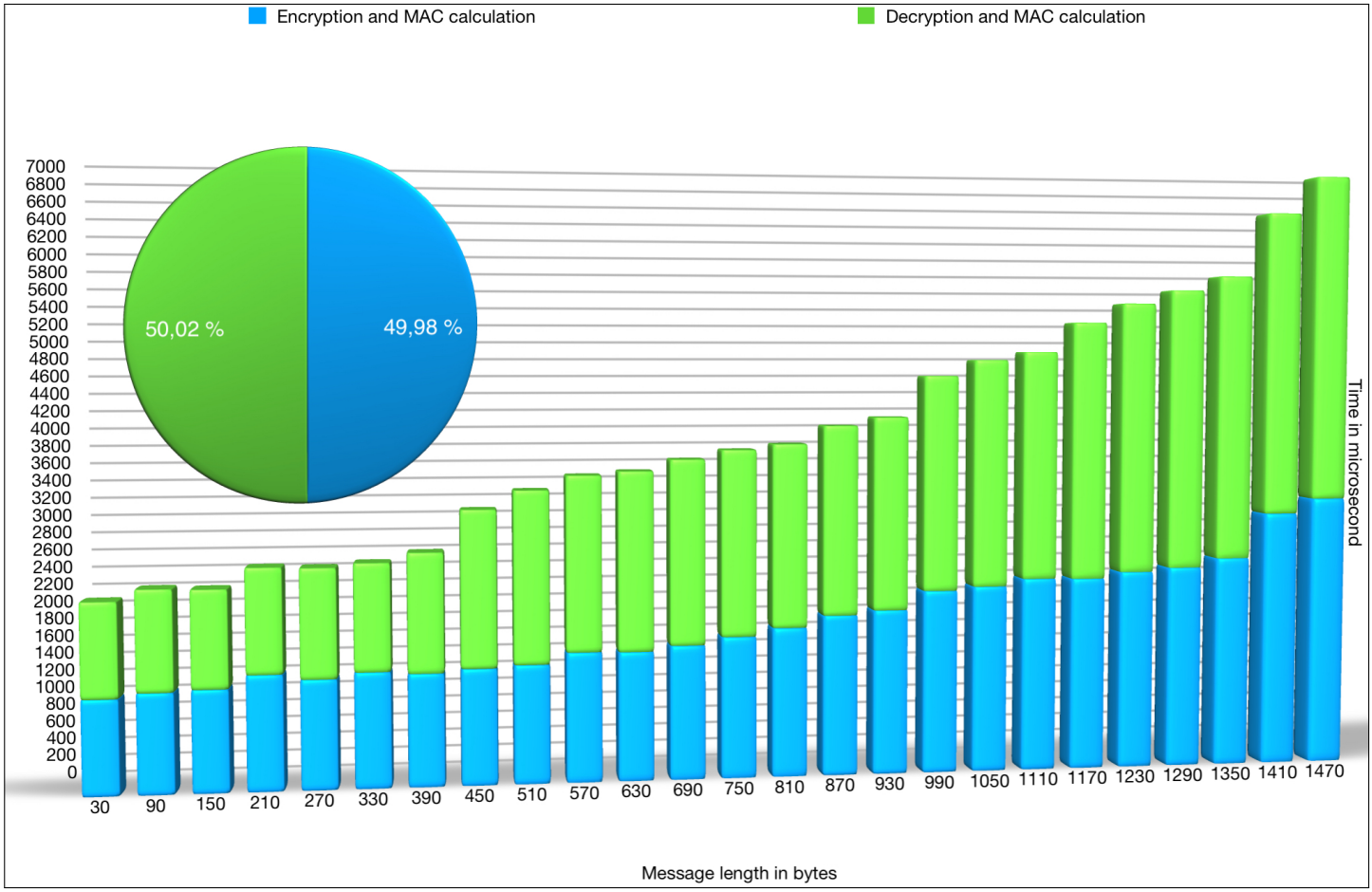

Fig. 5.16: *SafetyNETp after integrating the security layer*

The protected messages are encapsulated into DTLS frames then into the security layer frame. This introduces a data overhead as the secured messages contain the security layer and DTLS headers as well as the MAC. Furthermore, some encryption algorithms need to add padding to the messages. Figure 5.17 shows the data overhead.

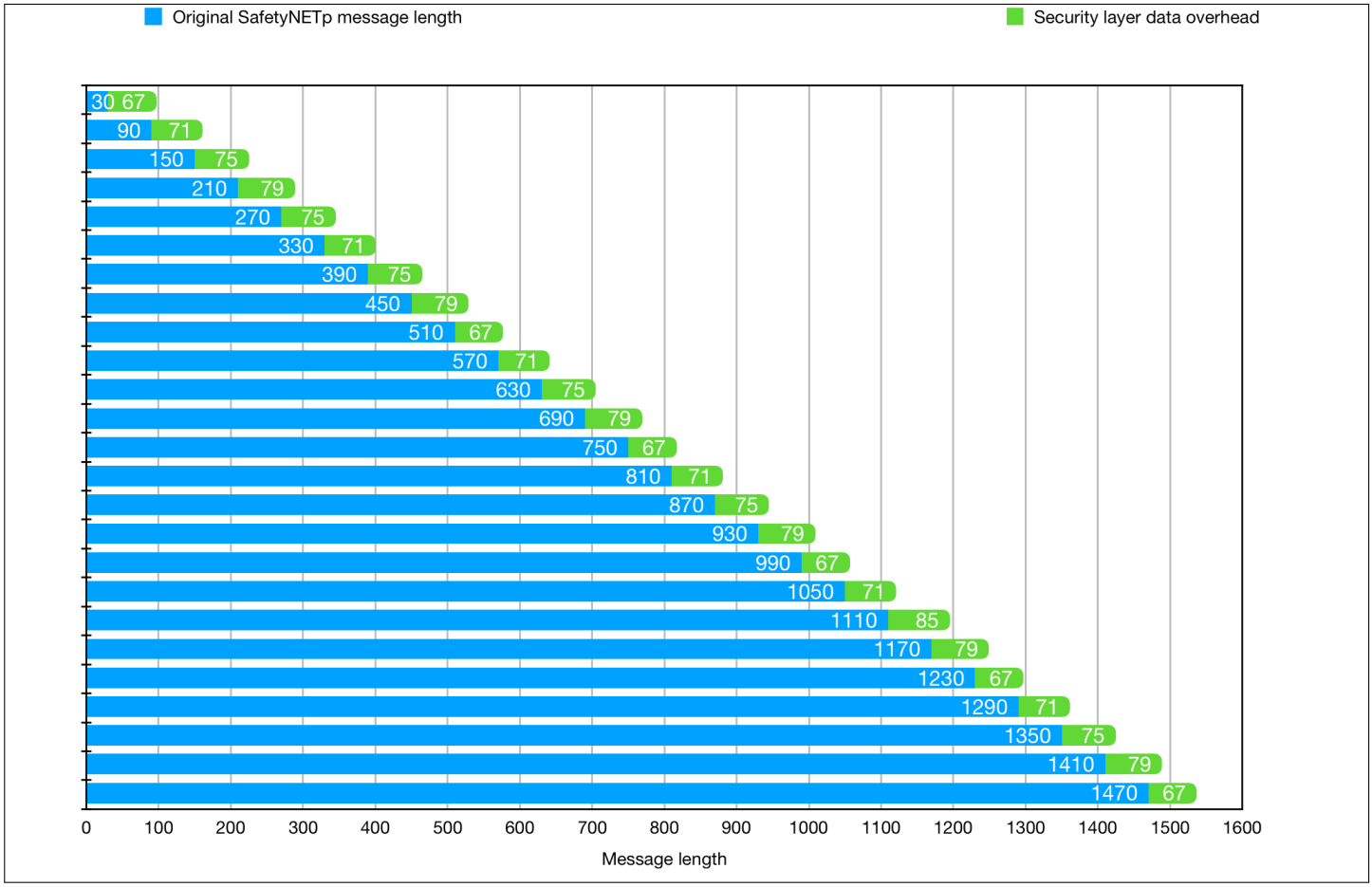

Fig. 5.17: *Data overhead*

The startup time of the system without our security layer is equal to the startup time of the SafetyNETp devices as the devices start exchanging *cyclic data* directly after the startup. However, with our security layer, after the startup, the devices establish the DTLS handshake before exchanging *cyclic data*. In fact, as shown in Figure 5.18 the time needed for the devices' startup is equal to *94 seconds*, the time needed to establish the DTLS handshake is equal to *7 seconds*. Therefore the new startup time is equal to *101 seconds*.

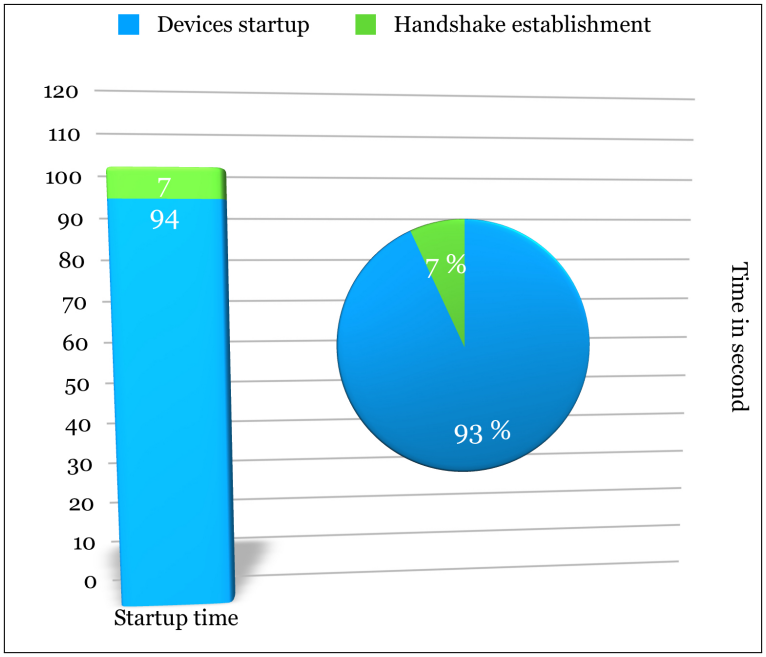

Fig. 5.18: *Startup time overhead*

Conclusion

Throughout this chapter, we presented the software and the hardware environments, we outlined the important implementation steps and we evaluated finally the performance of our solution.

General Conclusion and Perspectives

Today, most of the currently available industrial networks focus mainly on meeting real-time requirements and are designed and developed without any built-in security. SafetyNETp is one of the well known and most used Industrial Ethernet protocols, although it does not provide any special security measures. SafetyNETp based networks are vulnerable to attacks as it enables unauthorized access in a relatively straightforward manner given that all the communications are performed with no authentication, no integrity checks, and no encryption.

Designing and implementing a security layer for SafetyNETp from scratch is a difficult and very error-prone task. That's why we were interested in our project by the available protocols which are already tested and assessed by the internet community. DTLS and IPsec were the candidate protocols to be used. In fact, our protocol of choice was DTLS, this is mainly because DTLS is situated at the session layer whereas IPsec is situated at the network layer. Actually, this makes DTLS a lot easier to integrate.

The implemented DTLS based security layer provides confidentiality, integrity, and authenticity respectively by using encryption, MAC algorithms, and certificates. It is obvious that a time and a data overhead will be introduced by running those security mechanisms. In fact, the performance tests have shown that the time needed for sending and receiving a message over the security layer is around 7 milliseconds using TLS_RSA_WITH_AES_128_CBC_SHA256 cipher suite.

SafetyNETp provides two communication models compatible with each other which are RTFN and RTFL. RTFN is used when a cycle time over 1 millisecond is sufficient whereas RTFL provides a cycle time at microsecond level. The current design and implementation of the security layer concerns only RTFN and does not provide any security to RTFL. Therefore, the communication between the secured RTFN and RTFL is not possible. In fact, Securing RTFL needs entirely a new design, moreover, as RTFL time constraints are more tight, integrating a security layer may affect significantly its cycle time.

Bibliography

[1] Karl Koscher. *Experimental security analysis of a modern automobile*. 2010. URL: www.autosec.org/pubs/cars-oakland2010.pdf (cit. on p. 1).

[2] Richard Zurawski. *Industrial communication technology handbook*. 2014 (cit. on pp. 1, 3, 4, 7, 8, 12–14).

[3] T. Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Network Working Group, Aug. 2008. URL: www.tools.ietf.org/html/rfc5246 (cit. on pp. 2, 18, 19, 22, 23).

[4] E. Rescorla. *Datagram Transport Layer Security Version 1.2*. RFC 6347. Internet Engineering Task Force, Jan. 2012. URL: www.tools.ietf.org/html/rfc6347 (cit. on pp. 2, 22, 23).

[5] Safety Network International e. V. *SafetyNET p Real-time Ethernet for complete automation*. URL: http://donar.messe.de/exhibitor/hannovermesse/2016/C124313/safetynet-p-realtime-ethernet-eng-294559.pdf (cit. on p. 5).

[6] *Industrial Ethernet*. URL: https://www.kunbus.de/industrial-ethernet.html (cit. on pp. 9, 10).

[7] Profinet. *Industrial Ethernet for real-time applications*. 2013. URL: http://us.profinet.com/so-is-ethernet-deterministic-or-not/ (cit. on p. 10).

[8] *SAFETYNET P - FOCUS ON SAFE AUTOMATION*. URL: https://www.kunbus.de/safetynet.html (cit. on p. 12).

[9] Jonas Berge. *Introduction to Fieldbuses for Process Control*. URL: http://www.science.upm.ro/~traian/web_curs/Scada/docum/fieldbus.pdf (cit. on pp. 12, 13).

[10] *SSL/TLS History*. URL: https://www.acunetix.com/blog/articles/history-of-tls-ssl-part-2/ (cit. on p. 18).

[11] E. Rescorla. *Transport Layer Security Version 1.3*. Internet Engineering Task Force, 2018. URL: https://tools.ietf.org/html/draft-ietf-tls-tls13-28 (cit. on p. 18).

[12] E. Rescorla. *The Design and Implementation of Datagram TLS* (cit. on p. 22).

[13] E. Rescorla. *Datagram Transport Layer Security Version 1.3*. Internet Engineering Task Force, 2018. URL: https://tools.ietf.org/html/draft-ietf-tls-dtls13-28 (cit. on p. 22).

[14] *Pilz rail engineering*. URL: https://www.pilz.com/en-INT/products-solutions/industry/railway (cit. on pp. 28, 29).

[15] Henk C.A. van Tilborg. *FUNDAMENTALS OF CRYPTOLOGY*. URL: https://hyperelliptic.org/tanja/teaching/cryptoI13/cryptodict.pdf (cit. on p. 46).

[16] Kunbus. *Revolution Pi*. URL: https://revolution.kunbus.com/wp-content/uploads/manuell/datenblatt/revolution_pi_flyer_en.pdf (cit. on p. 51).

[17] *OpenSSL*. URL: www.openssl.org (cit. on p. 53).

[18] *WolfSSL*. URL: https://www.wolfssl.com/ (cit. on p. 53).

[19] *MatrixSSL*. URL: www.github.com/matrixssl/matrixssl (cit. on p. 53).

[20] *MatrixSSL*. URL: https://tls.mbed.org/ (cit. on p. 53).