1 2 UNIVERSIDADE D COIMBRA 9 0

Eduardo Martinho Martins de Andrade

SECURITY FOR 5G COMMUNICATIONS IN CRITICAL SYSTEMS

Dissertation in the context of the Master in Informatics Security, advised by Prof. Doutor António Jorge da Costa Granjal and Prof. Doutor João Paulo da Silva Machado Garcia Vilela and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra

June 2020

Abstract

The 5G technology will bring in a very close future, several new improvements to mobile communication systems, namely in terms of bandwidth, latency and scalability, as well as opening doors to new concepts and techniques. All of this makes 5G a serious candidate to be used in new areas and contexts where mobile communications are not usually used.

One of these contexts is the Critical Systems. Given it's strict performance requirements (very low latency, high number of nodes), mobile communications like 4G/LTE, were not a feasible solution on these environments, leading to the need of deploying dedicated closed networks, what comes with increased cost. Some of these areas are railway signalling and power distributions systems.

Shifting such systems to 5G networks means that systems usually using closed and dedicated transmissions systems will be using open ones. Moreover, current protocols used on such areas may not contain security mechanisms to prevent cyber attacks on open systems.

With this in mind, this research work addresses the security gap existing on protocols and critical systems, namely on railway signalling and power distribution, that the shift to 5G technology will create. We analyse the design of several solutions already being used, as well as other work already developed on security mechanisms to these systems. We propose, develop and evaluate a solution based on a bridging device and an expandable security library, containing several recommended security mechanisms for Critical Systems using the Power Distribution Systems as practical use-case.

This thesis is integrated in the 5G Mobilizer Project developed at the Centre for Informatics and Systems of the University of Coimbra, whose goal is to research, develop and validate a set of products running over 5G networks. The outcome of this work, will be integrated on the 5G Mobilizer project, more precisely on PPS3 - Products and Services for Machine-to-Machine communications, providing a security library and a bridging device implementation, combined to act as a Security Box to Critical Systems compliant with strict performance requirements.

Keywords

5G, Critical Systems, RaSTA, R-GOOSE, Railway Signalling, Power Distribution

This page is intentionally left blank.

Resumo

A tecnologia 5G irá trazer num futuro próximo várias melhorias para os sistemas de comunicação móvel, nomeadamente em termos de largura de banda, latência e escalabilidade, assim como abrir portas à utilização de novos conceitos e técnicas. Tudo isto faz do 5G um sério candidato a utilizar em novas áreas e contextos onde normalmente comunicações moveis não podem ser utilizadas.

Um destes contextos é o dos Sistemas Críticos. Dados os rigorosos requisitos de desempenho (latência muito baixa, elevado número de nós), redes móveis como o 4G/LTE, não eram uma solução válida para estes ambientes, levando à necessidade de criar redes fechadas e dedicadas, o que representa um aumento significativo do custo. Algumas destas áreas são os sistemas de sinalização ferroviária e os sistemas de distribuição de energia elétrica.

A mudança destes sistemas para redes 5G, significa que sistemas que por norma utilizam meios de comunicação fechados e dedicados, passem a utilizar meios abertos. Para além disso, vários protocolos utilizados por estes sistemas não foram projetados para meios abertos, pelo que não implementam mecanismos de segurança para prevenir ciber-ataques nestes termos.

Com isto em mente, este trabalho pretende investigar as lacunas de segurança que existem em tais protocolos e sistemas criticos, nomeadamente nos sistemas de sinalização ferroviária e sistemas de distribuição elétrica, que a mudança para a tecnologia 5G irá criar. Iremos analisar as diferentes soluções que são utilizadas atualmente, assim como outros trabalhos já desenvolvidos em prol da ciber-segurança nestas áreas. Propomos, desenvolvemos e avaliamos uma solução baseada num *bridging device* e numa biblioteca de segurança expansivel, contendo vários mecanismos de segurança recomendados para Sistemas Críticos, utilizando os Sistemas de Dístribuição de Energia como caso de estudo prático.

Esta tese está inserida no projeto Mobilizador 5G, a ser desenvolvido no Centro de Informática e Sistemas da Universidade de Coimbra, cujo objetivo está na investigação, desenvolvimento e validação de um conjunto de produtos que utilizarão redes 5G. O produto deste trabalho de investigação, será integrado no projeto Mobilizador 5G, mais precisamente, no contexto do PPS3 - Produtos e Serviços para comunicações Máquina-Máquina, fornecendo uma biblioteca de segurança, que poderá ser integrada diretamente em IEDs (Intelligent Electronic Devices), e um *bridging device*, sendo que em conjunto atuarão como uma *Security Box* para Sistemas Críticos, em conformidade com os seus rígidos requisitos de performance.

Palavras-Chave

5G, Sistemas Críticos, RaSTA, R-GOOSE, Sistemas de Sinalização Ferroviária, Sistemas de Distribuição de Energia Elétrica

This page is intentionally left blank.

Aknowledgments

First, I would like to thank my thesis advisors, Prof. Doutor Jorge Granjal and Prof. Doutor João Vilela, for accepting me in this project and for all of the help and support provided during this dissertation.

Secondly, I would like to thank to all of my friends that were always present when needed. I would like to specially thank André Reber for always being ready to help me, specially for all of the corrections and suggestions that he made.

Finally, I would like to give a special thank to my family. For all of the support on all moments, for all advises they gave me and for encouragement provided. Thank you for being tireless when I needed anything.

This work is supported by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Lisbon (POR LISBOA 2020) and the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project 5G with Nr. 024539 (POCI-01-0247-FEDER-024539)].

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

3DES Triple Data Encryption Standard. 20, 21

4G/LTE 4th Generation Long-Term Evolution. 1

BLE Bluetooth Low Energy. 24, 25, 33

CCC Command and Control Center. 4, 14, 15, 25

CENELEC European Committee for Electrotechnical Standardization. 26

CISUC Centre for Informatics and Systems of the University of Coimbra. 2

D2D Device-to-Device. 3

DNP3 Distributed Network Protocol 3. 13, 22–24, 30

EC Execution Counter. 19, 21

GOOSE Generic Object Oriented Substation Event. 21, 22, 38–40, 43, 48

IEC International Electrotechnical Commission. 26, 29

IED Intelligent Electronic Devices. 1, 6, 13, 22, 43, 48, 49, 55, 72

IoT Internet of Things. 15, 25

IP Internet Protocol. 11, 18, 21, 22, 25, 29, 30, 55

LC Level Crossing. 1, 3, 4, 10, 11, 17

LDT Linha de Desenvolvimento Tecnológico. 2–5, 15

M2H Machine-to-Human. 2

M2M Machine-to-Machine. 2

MAC Message Authentication Code. 19, 21, 24, 46–48, 59, 64

MEC Mobile Edge Computing. 4

MQTT Message Queuing Telemetry Transport. 15, 25, 33

NFV Network Function Virtualization. 1, 11, 12

PDU Protocol Data Unit. 18

PPDR Public Protection and Disaster Relief. 2, 4, 14, 15, 34

QoS Quality of Service. 14, 15

R-GOOSE	Routable-Generic Object Oriented Substation Events. 13, 21, 22, 28–31, 33–35, 38, 40–43, 45, 48, 49, 53–57, 59, 61, 63, 65, 67, 73–76, 86, 87
RaSTA	Rail Safe Transport Application. 11, 17–21, 33, 35, 37, 40
RBC	Radio Block Centre. 19
RSSP-II	Railway Signal Safety Protocol-II. 17, 20, 21
SCADA	Supervisory Control and Data Acquisition. 13, 22, 29, 38, 39
SDN	Software Defined Network. 1, 11, 38
SIL	Safety Integrity Level. 11, 16
TCP	Transmission Control Protocol. 19, 21, 29, 30
TLS	Transport Layer Security. 18, 19, 22, 25, 29, 30
TTS	Triple Time Stamp. 19, 21
UDP	User Datagram Protocol. 11, 18, 43, 55
V2X	Vehicle-to-Infrastructure. 3

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

In this chapter, we will introduce the document and give the context to the research work. Also, we will describe the 5G Mobilizer Project and its applications, namely the four development branches. Finally, we will present the goals we want to achieve, the work plan to both semesters and the document structure.

1.1 Context

The 5G technology will enter on the production phase by 2020, with the goal of replacing the 4th Generation Long-Term Evolution (4G/LTE). The technological developments provided by 5G will improve the overall mobile communications, in terms of performance (bandwidth, lantency, scalability, etc.) as also with the integration with concepts and techiques new to the area, such as Software Defined Network (SDN) and Network Function Virtualization (NFV). A very tempting step is taking advantage of those "new capabilities" and apply wireless communications on scenarios that usually cannot operate with the restrictions inherent to these networks, such as critical systems/applications and/or with an high number of components/devices connected to the network.

Nowadays, the systems that cannot lay under the limitations of the conventional mobile communications, mostly make use of cable networks, which implies a significant increase in the cost related to used material. On that scenario, the migration to a wireless environment is a common desire.

The railway signalling systems to Level Crossing (LC) are an example where the usage of wireless communication networks are quite interesting and promising. Those systems are composed by lightning signals, access conditioning barriers, sensors and system controllers, and they must all be interconnected. Also, the sensors and the controllers could be as far as 2km from each other, and usually they are connected by copper cables.

Other application scenario for the new mobile communication generation could be the electrical energy distribution. As a critical system, there are several strict requirements related to the quality of the communication between the components, plus, the high number of system nodes make it an even more interesting environment for 5G networks. The goal of these systems is to manage the distribution of electrical power over the grid, having in consideration that the system must be reliable and resilient to anomalous situations such as potential grid failures or intentional attacks. That system is composed by a large number of devices, continuously increasing, namely, Intelligent Electronic Devices (IED), substations and Reclosers. In order for the maximum quality of service to be achieved, it is really important that the controllers have access to as much information as possible, as for the best case, information from all system nodes.

On both presented scenarios, 5G will allow the integration of a wireless network architecture with an existing system in a critical scenario, taking care of the strict performance requirements and allowing high system scalability.

Nevertheless, mobile communications and in this particular case, 5G, will imply the usage of new and different techniques of system's monitoring and defence. On both presented cases, the security mechanisms already deployed and existing on the market, may not be enough, given the existence of new *players* and, therefore, we need to be aware of new threats and risks, making crucial the development of new solutions for these environments.

1.2 5G Mobilizer Project and Applications

This research work is part of 5G Mobilizer project [2], whose goal is the design, development and validation of products and solutions operating over the future 5G network, supplying new services or replacing the existing solutions being used by the industry. There are several parties working on 5G Mobilizer, such as the Centre for Informatics and Systems of the University of Coimbra (CISUC), where this work was developed. The project reaches all 5G applications contexts, being divided in different Products, Processes and Services (PPSs) with the given focus: Access Domain, Core and Vertical sectors with Machine-to-Machine (M2M) and Machine-to-Human (M2H) communications.

More precisely, this work was done on the context of PPS3, whose focus resides on development of products and services for M2M communications. Those products are:

- Self-healing solutions for monitoring electrical energy networks supported over 5G
- Critical solutions for railway signalling supported over 5G
- Public Protection and Disaster Relief (PPDR) platform solutions supported over 5G
- Intelligent video-vigilance in Ultra-High Mobility, using Edge Computing over 5G

There are several partners working in PPS3 namely University of Coimbra, EFACEC Energia, EFACEC Engenharia, OneSource, Ubiwhere and PDM&FC. Inside each development line of PPS3 presented, University of Coimbra is responsible for analysing and ensuring the networks security where each application will operate. In the next subsections will be presented and described the products with impact on this work.

1.2.1 Self-healing solutions for monitoring electrical energy networks

The goal of this "Linha de Desenvolvimento Tecnológico (LDT)" is the implementation and development of electrical power management systems with the capability to detect and react to faults and failures, in a quick, efficient and precise way, by means of self-healing mechanisms. Given the extension of the electrical power grid, as well as the under-laying network and several possible architectures for the self-healing solutions (centralized, semi-centralized, distributed), the implementation of this kind of systems might be very complex and expensive, since it is necessary to have a network able to support critical communications efficiently, having in consideration the large volume of components, as well as the constant growing of the network.

The introduction of 5G networks may satisfy those requirements, even if it is still important to have in mind the overall requirements inherent to the system, as very low latency and additional security measures.

The next figure (1.1) illustrates the network architecture that 5G Mobilizer wants to implement. There we can see all of the components connected over a 5G network, allowing the implementation of a fully distributed self-healing system.

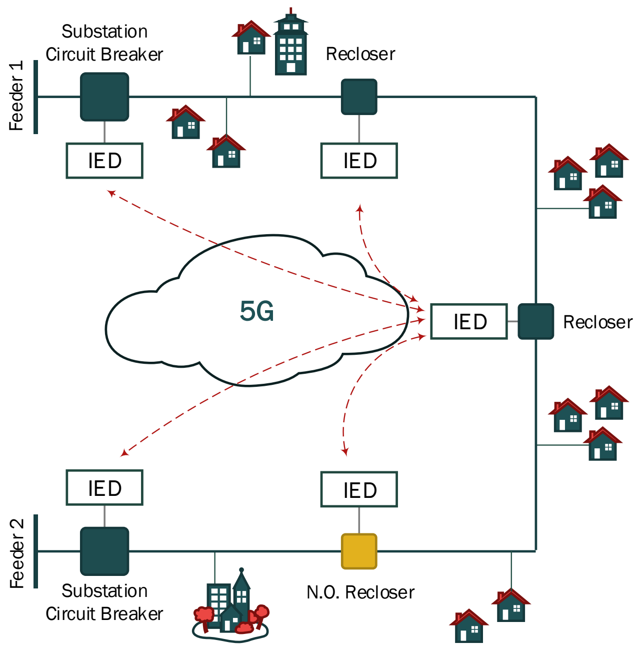


Figure 1.1: Eletrical energy network [47]

In this LDT the leading partner is EFACEC Energia. Their role is to implement the self-healing solution for power grid over the 5G network, while the role of University of Coimbra is to protect this system at the network level, on the 5G context and on the local systems.

1.2.2 Critical solutions for railway signalling

On the railway signalling environment, the usage of communication networks supported on public access infrastructures is not a common practice nowadays, given the security requirements and quality of service. However, this LDT intends to make use of 5G technology to support such systems, optimizing the closed time for LC points while satisfying strict requirements of the area.

The main goals are the development of:

- Announcement solutions over Device-to-Device (D2D) communications between detection triggers and access conditioning components controllers;
- Solutions for communications between vehicle (train), access conditioning barriers and signals controller (Vehicle-to-Infrastructure (V2X)), sharing information related to train positioning and respective velocity;
- Solution for communication between the LC and the train in order to exchange information and video-vigilance images from LC on crucial moments (train approximation to LC);

Figure (1.2) shows a typical railway signalling system, pointing out where 5G communications could be applied.

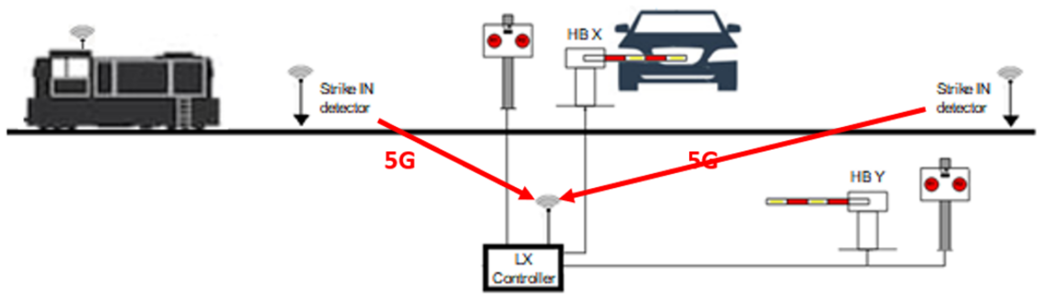


Figure 1.2: Railway Signalling scenario [47]

In this LDT the leading partner is EFACEC Engenharia. Their role is to implement the solutions for railway signalling systems over 5G, while the role of University of Coimbra is to ensure the security of such systems when operating over 5G networks.

1.2.3 PPDR platform solutions

The main objective of this LDT is the development of the product **BodyKit - Wearable Situation Awareness Platform**. This will be a device with several sensing capabilities (heart rate, fatigue levels, etc.) as well as other resources, as GPS, in order to give as much information as possible to Command and Control Center (CCC) about PPDR operationals on the field. The product must meet several strict requirements, mainly related to the security and reliability of communications, as a critical point for the system.

BodyKit will allow a better management and security of the resources, monitoring the whole operations scenario and all the means deployed on the field, from firefighters to police forces, and, using the sensing capabilities, will also be possible to monitor the operations environment, analysing temperature, radiation or chemical components. On Figure (1.3) is represented the BodyKit architecture, including its main components and communication flows.

In this LDT the leading partner is OneSource. Their role is to develop the BodyKit product. The role of University of Coimbra is to provide support in the security area that the usage of 5G networks will arise.

1.2.4 Intelligent video-vigilance in Ultra-High Mobility using Edge Computing

The fourth LDT has as main goal the development of a solution for optimizing the network utilization of an operator, and provide to the clients a set of functionalities associated to video surveillance, independently from the camera provider, as for example, a set of advanced statistics and analytics, detection of anomalous events and automatic images quality differentiation to the monitoring and controlling center, based on a Mobile Edge Computing (MEC) architecture. On Figure (1.4) is represented part of system's architecture, more specifically communication links existing between components.

We should also highlight that the product developed by this LDT will be used on the use case of intelligent video surveillance and transmission from LDT2, to detect obstacles at LC.

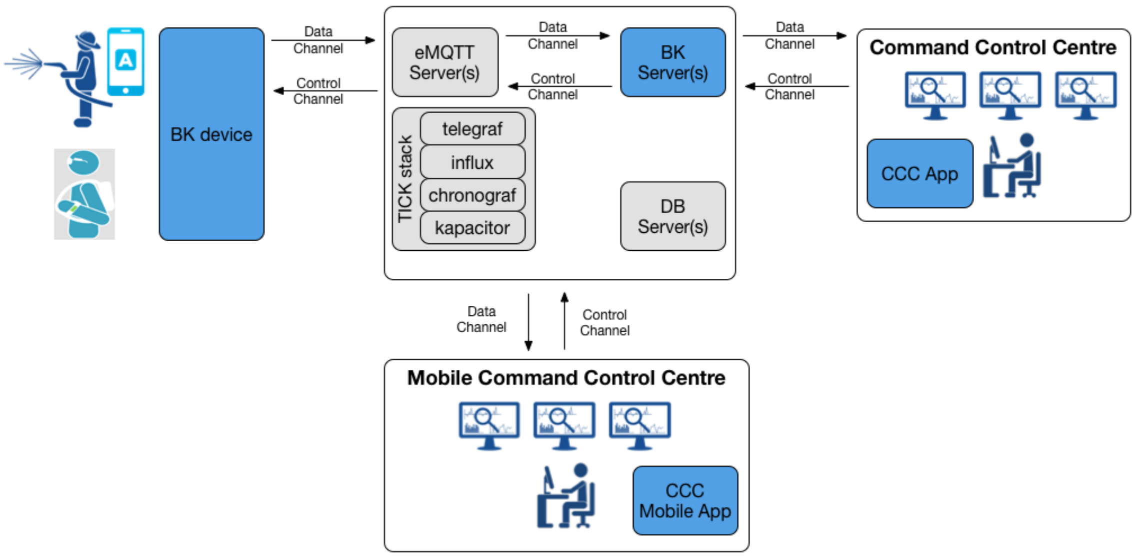


Figure 1.3: BodyKit Architecture [47]

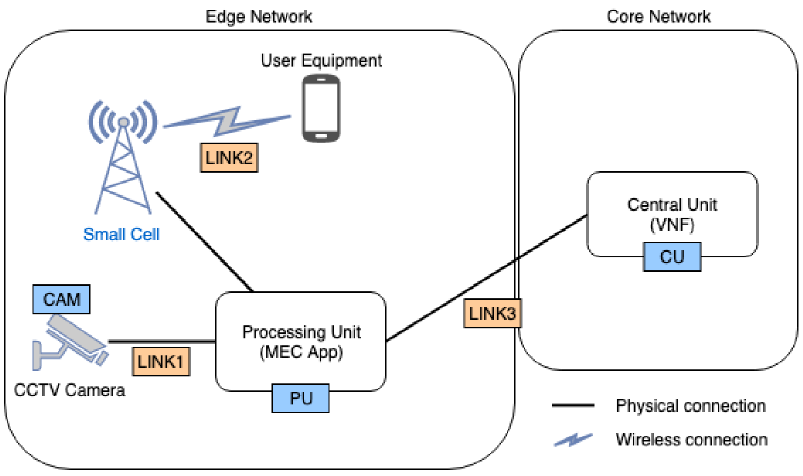


Figure 1.4: LDT 4 Architecture [47]

In this LDT the leading partner is Ubiwhere that is responsible for the development of the proposed solution. At the same time, all of the other project partners will provide support in the development, and more precisely University of Coimbra will provide support on the security of the networks and systems.

1.3 Contributions

This research work is part of the 5G Mobilizer project and several contributions were made to it, more specifically on the context of the work done at PPS3.

More precisely, the main contributions were:

- Identification of security requirements for the applications on the context of PPS3 (application specific requirements) and identification of transversal PPS3 application requirements, as well to the general critical applications that can be supported over 5G networks, although focusing on highly critical applications, such as Railway Signalling and Power Distribution systems

- Design of a security solution transversal to all PPS3 applications composed by a security library and bridging device
- Development of a security library containing a set of security mechanisms that can be applied directly on dedicated devices (IEDs) or used as a standalone solution
- Development of a bridging device to support the security library standalone version
- Assessment and validation of the developed mechanisms, taking in consideration the security requirements, robustness and performance required by the applications and systems

This is an ongoing work in the context of the 5G Mobilizer project, thus the outcome product of this research work is being integrated in a larger testbed along side with other products. In this testbed, our work will be placed on the context of Power Distribution Systems, acting as a security gateway for R-GOOSE communications providing security mechanisms to legacy IEDs with less computational capabilities, and will also be included directly on more capable IEDs.

1.4 Work Plan

The work plan for the 1st Semester were focused on studying and analysing the system inside the scope of the work, the state of the art and related work. The main activities performed on 1st Semester were the following:

- Acquire background knowledge
- Study State of the Art
- Perform Gap Analysis
- Write dissertation document

The next image illustrates the work done on 1st Semester:

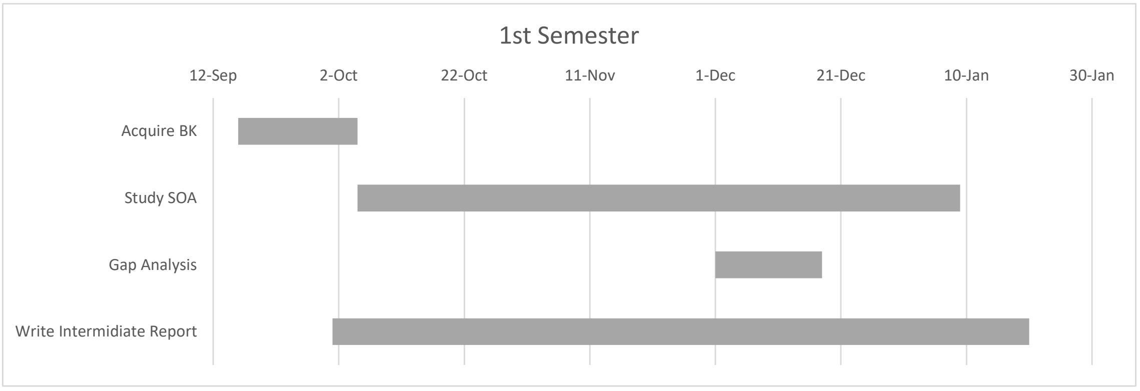


Figure 1.5: Work plan 1st Semester

The work during the 2nd Semester were focused on develop and evaluate a set of security mechanisms and all of the other components of our solution. Given that, the main activities performed during the 2nd Semester were the following:

- Implement a baseline environment (testbed without security mechanisms)
- Develop new security mechanisms or changes to protocol implementations
- Evaluate proposed mechanisms and analysing results
- Write dissertation document

The next image illustrates the work planned to the 2nd Semester:

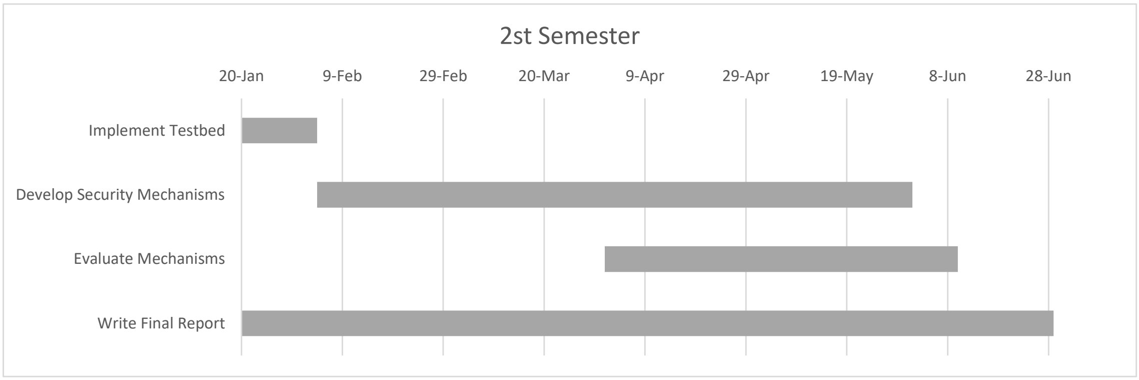


Figure 1.6: Work plan 2st Semester

1.5 Document Structure

The document is structured as follows. Chapter 2 presents some concepts and background knowledge that was necessary to acquire, mainly related to the areas of focus. Chapter 3 is the State of the Art, where we discuss current solutions used on Railway Signalling, Power Distribution and PPDR Systems, we establish a set of requirements for each application and we perform a gap analysis between those requirements and the security mechanisms already deployed in each system. In Chapter 4 we propose our solution and we describe its implementation. The fifth chapter is the Results and Analysis, were we present the functional and performance evaluation done. Finally, Chapter 6 is the Conclusion and Future Work.

This page is intentionally left blank.

Chapter 2

Background Knowledge

In this chapter, some concepts and introductory topics will be explored, in order to establish an important support base for the following phases. The critical systems topic will be addressed, giving a possible definition and the different categories into which they are subdivided. Then, will be done an introduction to the different scenarios and contexts addressed during this work, such as railway signalling and power distribution grids.

2.1 Critical Systems

Critical systems are those that, in the event of a failure, can directly result in threats to human life or an organization, also possibly leading to major economic and/or environmental losses. We can further classify critical systems into three sub-classes: **Safety Critical**, **Mission Critical** e **Business Critical**:

- **Safety Critical** - Systems operating in scenarios that may lead to loss of human life, serious injury or damage to the environment.
- **Mission Critical** - Systems that must assure that a certain goal or function are properly performed, regardless of a component failure or inoperability.
- **Business Critical** - Systems that are directly related with the goal of the organizaion's business, and, when compromised, may cause economic costs or affect other business indicators, such as its reputation.

For each of these system categories, there are properties that must be guaranteed, although it may vary with the scenario and system. It is also important to mention that, given the potential consequences in the event of a failure in these systems, there are strict rules and laws to ensure a minimum level of confidence on the security and safety of the systems, which will be covered in the **State of the Art** section of this document.

2.2 Railway Signalling

Nowadays, railways are still widely disseminated, with large usage in European Union and other countries like United States of America, China, Germany, Russia or India. Even in Portugal, railways remain as a main public transportation, with almost the entire country being covered by the network. As a public transportation mean, it is intended to move a large number of users (for example, an Alfa Pendular (Portuguese high-velocity train) has around 300 seats), meaning that an accident will endanger a lot of people. Furthermore, the environment where it operates (heavy vehicle, with steel wheels, running on a steel rail, which leads to a big breaking distance) highly increments the probability of an accident.

Given that, several techniques emerged to control and monitor railways, such as tracks segmentation and railway signalling. Track segmentation is the division of the railway into smaller parts, not just in the portions between stations, but also dividing those on smaller segments. On those segments it was easier to apply signalling, and, for example, it could be established that there could only be one train per segment. Naturally, signalling methodologies and techniques have been evolved from fixed blocks to moving blocks and train footprints, allowing a better management of the overall system. Railway signalling is used in various situations and its goal is to give information to train drivers related to the obstruction or possibility to keep moving forward. Some of those situations are the circulation between track segments and the intersection of the railways with roads, also called Level Crossing (LC).

For the blocks, the signalling system needs to assure that in each segment (or block), there is, at most, one train at the same time, and that, in case of a failure, the following train can safely break without crashing. The Figure (2.1) shows an example of the signalling system operating between segments (blocks) of the track, where it is established that the safe distance between trains is of two blocks, both represented by the yellow signalling lights. The green lighted blocks are the ones where a train could travel to, while the red ones, are prohibited, due to the presence of another train.

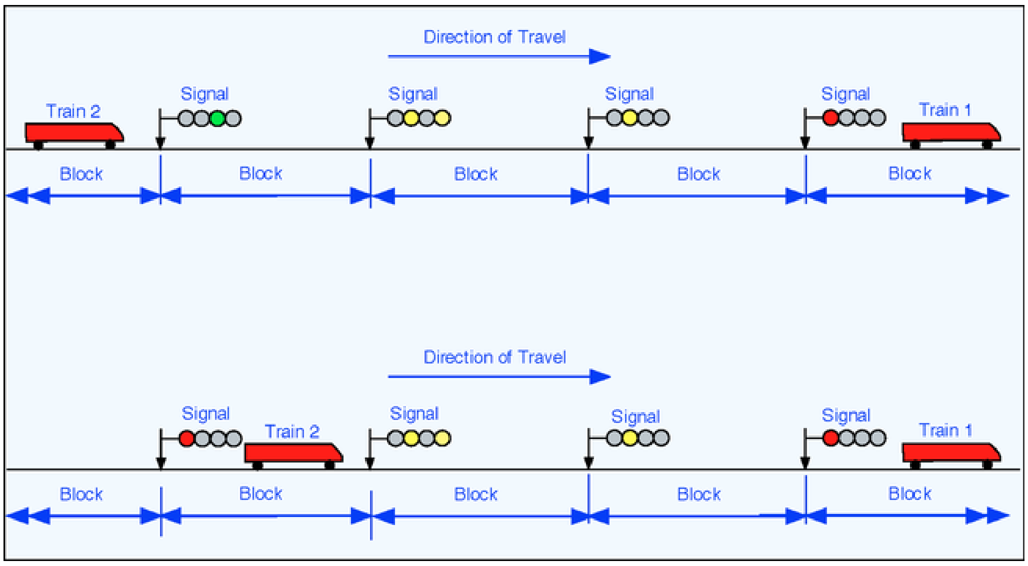


Figure 2.1: Railway signalling operating on segments context [57]

As for the LCs, there are other factors that must be taken into consideration. Inherent to LCs are the cars crossing, which represent a different safety threat, as they are not so predictable as other trains. From the beginning until now, we stated that LCs are also a critical point for public safety, as it is also a major point from pedestrian accidents. With safety as a major concern, it is also important to optimize the closing time of LCs as it can also prevent risky movements from pedestrians and road vehicles.

2.2.1 System Overview

Railway signalling systems have been suffering several changes since the very beginning of railway: from the non existence of signals at all, to the full automation of these systems. They are composed by several components and devices: activating triggers, light signals, access conditioning barriers, audible signals and LC controllers.

The **activating triggers** (or train sensors) have the function of detecting the passage of a train, notifying the controller and other components that a new train is about to pass through. These triggers are placed on the rails and are activated by train contact, representing a sensing component to the system. The triggers are a crucial part of the system, if they fail to detect the train, the entire safety of the system and their users is compromised.

The **light signals**, the **access conditioning barriers** and the **audible signals** are the actuators of the system. That means that they respond with an action, when they receive some kind of message or signal. Their function is to physically deny the access of road vehicles or pedestrians to the intersection.

Finally, the **LC Controller** is the "mind" of the LC. It Is the component responsible for managing the LC and controlling all the other system components, receiving data from triggers and sending signals to actuators. This kind of component are specifically developed for this function by several companies such as Siemens. Nowadays, these controllers are not limited to respond linearly to triggers, as they try to optimize the closed time of the LC. The controllers are constantly measuring the distance between the train and LC, and with the information from the LC itself, they can determine the optimal moment to close the barriers. However, this method cannot be applied in every LC, as they usually use reflectometry, and also, significantly increase the overall cost of the system.

In order for the entire system to work properly, it is necessary the existence of a communication network that allows the multiple components to exchange data. The network itself is usually simple, connecting all the components to the LC controller.

The network connections are usually made by copper cable, supporting normal network protocol stack architectures as Ethernet for the Link Layer (or OSI layer 1-2), Internet Protocol (IP) on Internet Layer (or OSI layer 3) and User Datagram Protocol (UDP) on Transport Layer (or OSI layer 4), taking in consideration the TCP/IP Protocol Stack. On top of those base protocols, exists others to ensure some system properties, such as safety, reliability or security. A widely spread example of such protocols is Rail Safe Transport Application (RaSTA), being another transport protocol tailored specifically to meet the strict requirements of railway signalling systems. RaSTA represents an additional safety layer that must be included on protocol stack for railway signalling systems, in order to ensure that safety-related requirements and Safety Integrity Level (SIL) is achieved. In the Figure (2.2) is represented the railway signalling system existing in a LC. There we can see the triggers, several actuators (light signals and barriers), the controller and the cable based network that connects all the components.

The fact that those networks are supported over copper cables represent an obstacle in several ways: it is expensive to deploy such network because sometimes the distance between triggers and the controller can reach up to 2km; the time it takes to deploy such a network and the working cost is also expensive, as it involves a lot of work; that kind of network makes it difficult to deploy new and innovative components, as well as it doesn't support interesting new network features such as SDN and NFV. Software Defined Network (SDN) is a network architecture that uses programmable software-based controllers to manage traffic and communicate with hardware infrastructure. This provides a centralized view of the network, allowing an holistic network management and more granular security. Network Function Virtualization (NFV) allows the

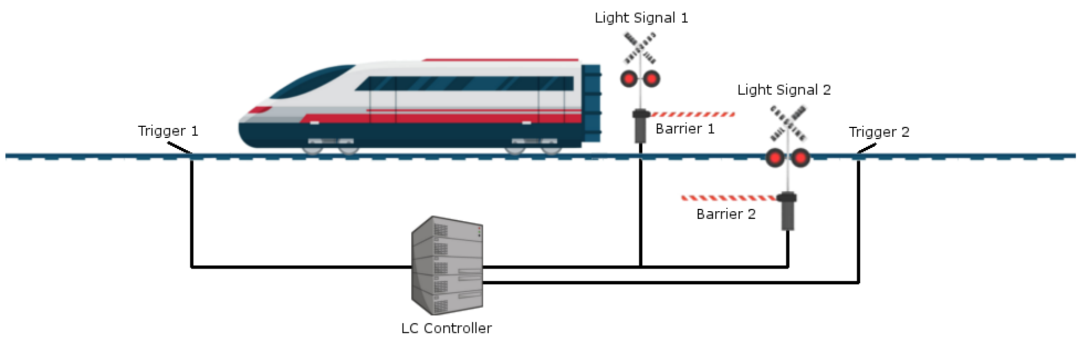


Figure 2.2: Railway Signalling System diagram

virtualization of common network functions, reducing the cost of deploying new components to the network. As an example, using NFV, it is not necessary to buy new dedicated and expensive hardware components to add specific network functions. Instead, it is possible to use a common server or switch and deploy a software version of that network function.

An important aspect to point out is the classification of the environment where these systems operate. The occurrence of a system failure, in most cases, results in serious danger situations and accidents. Those accidents will probably endanger the human lives and the environment itself. Having this in consideration, we can characterize these systems as ***safety-critical***. As mentioned before, critical systems must comply with more strict rules and regulation, thus, there are several standards they must address. The most important standards will be described and discussed later in the **State of the Art** section.

2.3 Power Distribution

Power distribution systems represent a major and important infrastructure of the society since its very early days, and now is even more deeply connected to the way we manage our lives. We probably wouldn't be able to do anything we do in a normal day, if entire power distribution network goes offline.

As every new invention, electric energy distribution started with small grids and lines, and not computer managed as it is today. However, as the grids and voltage rates started to evolve and increase, there was obviously the need of automation on this environment. Nowadays, these systems are responsible for one main function, receiving the generated electrical energy on the entry point of the grid, and transmit it to the end-users, as for example, normal citizens or companies. Inherent to their mission, this systems must assure that power distribution is safely done, as well as assuring that all the clients are properly supplied.

In order to achieve that, these systems are composed by various components and several points of management. Nowadays, the electrical grid is huge, with a lot of endpoints, so it is divided in multiple sections, managed by substations [14]. This division allows the grid managers to properly understand each section and also to easily identify the occurrence of failures. A failure on these systems can be a problem in one of the transmission lines, meaning that some section or part of a section of the grid might be "on the dark". A simple way used to resolved this kind of problems was the construction of the grid in a ring "shape". Basically, grids usually are built in ring shapes, meaning that they are a closed network, or that every node network has at least two distinct paths to be reached. That way, if one of these paths has a problem, the station can use the other.

However, managing this kind of systems is very complex, because it is constantly growing and already has a huge amount of nodes. Some techniques used are based on self-healing, where the grid has the ability of identifying where the failure occurred and deviate the power transmission to a redundant line, covering the previous node. An actual challenge is related to the way this self-healing capability is achieved, as some techniques, more precisely the distributed self-healing, needs that various endpoint nodes have the ability to communicate with each other, which is impossible with the technologies used on the field today. This challenge is one of the main applications addressed by 5G Mobilizer Project.

2.3.1 System Overview

Given the importance of these systems and the large environment where they operate, they are usually very complex, including a large amount of components. They are composed by power generation stations, power substations, medium voltage power transmission lines and distributions lines, that connect several individual end-consumers. Nowadays, such systems use dedicated networks to be managed.

Other important component are IEDs. These are microprocessors-based controllers of power system equipments, and they operate by collecting data from sensors and issuing commands to other system components, as well as raising events, such as signalling a failure. These components, are of major importance as they allow the grid to be managed and to be supervised.

As for the mechanisms to recover from failures, self-healing mechanisms are usually used. Self-healing of the grid can be achieved by various architectures, as it can be centralized, distributed at substation level and distributed at feeder level. Different solutions have different advantages and also limitations. In the first one, the centralized architecture, self-healing is controlled on the central station, meaning that it is highly scalable, although this creates a single point of failure on the central station. If a failure occurs there, managers will not be able to "see" that entire section, as well as it will not be able to recover from a potential failure on the grid itself. Normally, this scenario uses a Supervisory Control and Data Acquisition (SCADA) or similar scheme. On a semi-distributed architecture, the self-healing is coordinated from several substations, meaning that if a failure occurs on one substation, only that section will be off, and the others won't be affected. However, such scheme is not so scalable as the last one, because there is also the need of deploying new dedicated links. For the last architecture, the self-healing is done by coordinating several IEDs, and they will identify and recover from the failure by themselves. This eliminates the single points of failure, but tremendously increases the complexity and also requires the usage of modern IEDs. On the context of the quality of service provided, this is a great solution, as it represents the lower response time to a failure, given the proximity to the field of the recovery mechanism.

All of those components are connected over dedicated networks, either inside the same substation (connecting the local IEDs) or interconnecting the substation's networks. These networks usually support normal network protocol stack architectures as Ethernet-IP-TCP/UDP and finally the upper level protocols. At the upper level, the protocols are designed to ensure the high availability, resilience and reliability that this system requires. Some examples of such protocols are R-GOOSE and DNP3.

As we can see, this industry has a huge impact on today's society, being one major infrastructure that must be always working properly. In the event of a major fault the probability of great impact with severe consequences sharply increases. As an example, we can imagine that, if the entire power distribution grid goes offline, there exists the possibility that hospitals, air traffic controllers, emergency response services and other critical services couldn't operate, meaning that a huge amount of lives and also the environment will be highly endangered. At the same time, from the perspective of power distribution provider, a major fault will certainly endanger their ability to achieve their mission, providing electrical energy to those who need it. Given that, there are several Quality of Service (QoS) aspects that must be taken in consideration, such as communication latency and security, being the aim of this research work. So, another function that these systems must provide is the management of potential failures in the grid, as well as the capacity to recover. From this analysis, we can categorize this system as critical, more precisely, as **safety-critical** and **mission-critical** system.

2.4 PPDR Platforms

Public Protection and Disaster Relief (PPDR) agencies play a very import role on society. From the Public Protection perspective, they are intended to maintain law and order, protecting life and property, as well as responding to emergencies. Although, their objective is also responding to disasters and situations that endanger or pose as a dangerous threat to human life, the society itself, or the environment. With the increasing number of public safety related events [54], their functions are more critical than ever.

Usually, when a disaster occurs and PPDR Agencies are called to operate, there is the Command and Control Center (CCC) from where all of the operations are controlled. Thus, the system that allows the communications and information exchange between CCC and the operationals on the field, represents one of the most important assets.

Nowadays, PPDR systems rely on old and well tested protocols and solutions, because they offer an high level of confidence and resilience. However, it has been proven that it is important that such systems evolve in order to improve the quality of the service provided. As these systems tend to be old, they do not easily allow the integration with new technologies, as well as the extension of the current services. They mainly rely on voice communication, and radio transmission systems, using protocols that were not designed for transmitting other types of data, such as video.

Some of these PPDR solutions are TETRA, APCO P25 and DMR. TETRA Protocol stands for Trans-European Trunked Radio, and it is a European standard for a trunked radio system that was specifically developed for PPDR agencies. This is widely used in Europe, although it only provides mission critical voice communications, SMS and low speed data transfers. APCO P25 is the corresponding standard to TETRA in America, and it was also designed specifically for PPDR applications, providing mission critical voice communications, SMS and low speed data trsnafer. DMR stands for Digital Mobile Radio and it is a standard defined by the European Telecommunications Standards Institute. It is also used as a PPDR solution due to its lower cost. All of that solutions lack on the capacity of transmitting general data with low latency, as for example, sensorial data from operationals.

With this in mind, evolving such systems represents an important challenge to improve the quality of the service provided by these agencies, providing new systems capable of transmitting in a fast and efficient way new types of data to the CCC. One of the main goals of PPS3 of 5G Mobilizer project, is to develop **BodyKit Platform**, that will provide and integrate new functionalities, namely in terms of new sensing capabilities, as well as video transmission, to existing PPDR platforms, using new technologies as 5G to achieve the strict requirements of these environments.

2.4.1 System Overview

LDT3 of 5G Mobilizer project is responsible for the development of **BodyKit** product. BodyKit is a platform to provide new capabilities to PPDR agencies, using 5G technology to enhance overall system performance, as well as allowing the integration on new functions, such as real-time video transmission from field agents.

This product has an IoT based architecture, considering the integration of multiple low-capacity sensing devices to collect data in a scalable way. The main objective is to provide a *full situation awareness* to the CCC, including real-time video transmission, location and sensing data from field agents.

The system is composed by three major components:

- **BodyKit Device**
- **BodyKit Servers**
- **CCC Application**

The BodyKit device is then composed by two other components, the wearable device with the sensors, and an application running on a smartphone, collecting data from the wearable device and sending it to servers. The wearable device has several sensors divided in two categories: biosensors, responsible for collecting data related to the field agent, and environmental sensors, responsible for collecting data from the surrounding environment. The biosensors present in BodyKit are the following: *electrocardiogram*, *respiratory rate*, *blood pressure*, *heart rate* and *body temperature*. It is then equipped with the following environmental sensors: *gas detection*, *smoke detection*, *humidity*, *environmental temperature* and *GPS*. Also, BodyKit allows video and audio transmission and includes an emergency button, in order for the agent to request assistance in case of an emergency.

The system is then composed by several servers to manage all of the information flows. The BodyKit Server is responsible for managing the communications between other servers and the CCC application, as well as responsible for realizing heavy computing operations that low-capacity wearable devices and smartphones are not capable of. More detailed, it has to do some processing related to event detecting, as for example, stress conditions of agents to trigger an alarm on CCC. Moreover, it configures the QoS of the network, because it might be necessary to prioritize some data flows. Finally, it is also responsible for part of the authorization and authentication procedures for the communications. Other important server is the MQTT Server. This is responsible for receiving data from BodyKit Device and managing the data exchanged, while enforcing authentication of users and also providing access control lists. Under all of these servers, there is a database to store several configurations, security policies and various mission related information.

As we can see, there are several means of communication on the system, with the most important being the communication channel between the wearable device and the smartphone, and the communication channel between the smartphone and the BodyKit Servers, namely the MQTT Server. In the 5G Project, for the communication between the wearable device and the smartphone, it is going to be used Bluetooth technology, more precisely, Bluetooth Low Energy, given the proximity of both devices. The communication between the smartphone and the MQTT Server will be done over 5G, using the MQTT protocol. A detailed analysis of such protocols will be done in the **State of the Art** chapter.

2.5 Safety Integrity Levels

On critical systems, specially safety-critical systems, a given function must be done taking in consideration safety, meaning that the probability of the system entering on an unsafe failure state must be minimal. However, failures and threats have a given risk, that is determined by its probability of occurrence and the impact it has, and are also associate with a given function or component. Given that, it is important that, in a safety-critical environment, such systems enforce safety, possibly by including a safety layer.

Various standards, such as **IEC 61508** [32], defined that such safety layer should exist and that such layer must decrease the safety risk by a certain magnitude, that is related to the operating environment. That magnitude, or level, is related to the SIL of the system. That magnitude is also related to the importance of the function to be done, as well as the impact that a unsafe failure could have.

On that standard are defined different SILs, for continuous or high-demand functions, based on their mode of operation, and the SIL is determined by assessing the probability of a failure occurring. The identification of the SIL that a given system needs is highly important, not only because in such environments almost every time it is demanded by regulations and laws, but as well as for developers and managers, which represents a major help and source of confidence on the system they developed or are managing. On the next tables are presented the SILs for both continuous and low-demand modes, as well as the probability of occurring a failure.

Safety Integrity Level	Probability of Occurring a Failure
1	$\geq 10^{-6}$ to $< 10^{-5}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
4	$\geq 10^{-9}$ to $< 10^{-8}$

Table 2.1: Safety Integrity Levels on Continuous/High-demand Operation mode systems [32]

Safety Integrity Level	Probability of Occurring a Failure
1	$\geq 10^{-2}$ to $< 10^{-1}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
4	$\geq 10^{-5}$ to $< 10^{-4}$

Table 2.2: Safety Integrity Levels on Low-demand Operation mode systems [32]

2.6 Chapter Wrap-up

In this chapter we introduced several topics that are going to be important through this research work. More precisely, we characterized Critical Systems as systems that in a event of a failure, can endanger human life, the environment, its objective or the business itself. We also discussed Railway Signalling and Power Distribution, giving a brief introduction on its main objectives and how they are achieved. Finally, we analysed the concept of Safety Integrity Level and its relation with safety-critical systems. On the next chapter, we will focus on analysing the communication protocols used by these systems, the standards they must comply with, and researching for related work on these areas. From the standards, we will define a set of requirements that each system must comply with.

Chapter 3

State of the Art

In this chapter, we will focus on several important topics that will be approached during this work. Firstly, we will analyse the communication protocols used on the 5G Project PPS3 use cases, namely railway signalling systems, power distribution systems and PPDR platforms, understanding the security properties each one provides. Secondly, we will focus on the requirements of each application, either in terms of security or performance, having in consideration the new open and wireless network scenario, the 5G. Then, using the conclusions from the previous sections, we will make a gap analysis between the security properties that the communication protocols already provide, and the requirements that these systems need to comply when used over 5G. Finally, we will present a review of several works done on this area.

3.1 Communication Protocols

In this section we will present and analyse several communication protocols that are used by the industry on the three applications. All of these protocols were designed specifically to be used on critical systems, thus, they are focused on assuring the reliability of the communications, tending to not address security.

3.1.1 Railway Signalling Systems

On the railway signalling systems, the communication protocol is responsible to exchange messages between the triggers, the LC Controller and the actuators, ensuring that the messages are properly delivered withing a strict time frame. There are several network protocols that could ensure the reliability of the communications, although only a few are tailored to operate under extremely strict time restrictions.

Some of them are **RaSTA**, **UNISIG SUBSET-098** and **RSSP-II**. On the next subsections, we will describe in more detail each one of these protocols, focusing on the safety and security properties they provide.

RaSTA

Rail Safe Transport Application (RaSTA) is a network transport protocol, designed specifically to railway signalling systems. As so, it was developed to achieve and assure the properties inherent to safety-critical systems, as reliability and resilience related to the network communications.

Main objective of RaSTA is to provide a reliable connection in a safety-critical environment between two endpoint components.

RaSTA is composed by two layers, inserted between the Application Layer and the Transport Layer, being **Safety and Retransmission Layer** and **Redundancy Layer**, as illustrated in Figure 3.1. Usually, RaSTA is used over UDP and Ethernet, having in consideration the normal network composition. These two layers are intended to ensure transmission system requirements defined on EN 50159 [19]. RaSTA guarantees all four requirements: **message sequence**, **message timeliness**, **message integrity** and **message authenticity**.

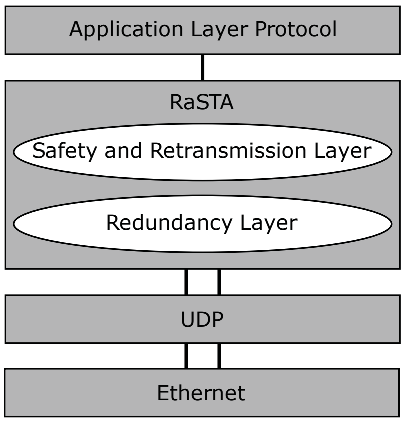


Figure 3.1: RaSTA Protocol stack [28]

From a top-down perspective, when a RaSTA message is sent, the Safety and Retransmission Layer is responsible for identifying the clients (system components), inserting the sequence numbers on messages, timestamps and a checksum. The clients identification is usually made by number ID which, with other mechanisms, assures the authenticity of the message. The usage of sequence numbers, mandatory by EN 50159, has the objective of identifying invalid packets (forged or errors), as well as monitoring the reception of those packets by the client. In case of a missing "acknowledgement" from the client relatively to a given sequence number, this layer retransmit that message. Timestamps enforce the invalidation of old and outdated messages, that shouldn't be considered anymore. The message integrity is achieved by the checksum, *safety-code*, added to the message payload, allowing the receiver to validate and confirm it. Ultimately, this layer is constantly monitoring the connection, using heartbeat messages. The second layer, Redundancy Layer, has the responsibility of transmitting the message over redundant channels, for example, over distinct UDP channels. These redundant channels should be physically distinct in order to provide real transmission redundancy. After the encapsulation of the message on this RaSTA data units, it is encapsulated on the remaining Protocol Data Unit (PDU)s, such as UDP and IP.

More precisely, RaSTAs *safety-code* is an 8B checksum based on MD4 hash function. It uses all the fields of the RaSTAs PDU to generate the hash. It is also with the safety-code that RaSTA allows network separation, by changing the initial vector of MD4 hash function. Using different initial vectors it is possible to separate RaSTA traffics within the same physical channel, as the applications will only validate packets generated with their initial vector.

From the security point of view, RaSTA allows three major security mechanisms: message hashing, TLS and IPSec. Message hashing is achieved with the safety code, as mentioned previously. More detailed, RaSTA hashes the entire PDU, containing the headers and payload, with MD4 algorithm, resulting on 16B, being then truncated into an 8B checksum. This provides message integrity, however, the usage of MD4 algorithm also represents a risk, as it is already considered a weak hashing algorithm, as analysed in [28]. Other major problem on RaSTA security, specially when moving from a closed and isolated network to an open access network, is the way how MD4 algorithm is applied. RaSTA allows separation of different messages flows, for example, "networks" from different railway stations. This is done, changing the initialization vector of MD4 based on the network key. This system provides some degree of overall security, however, if an attacker gains access to such key, it is able to forge valid messages, endangering the authenticity of system messages. In [28] multiple attacks are described to explore such functionality, demonstrating the weaknesses of MD4, also proposing the usage of other hashing algorithms as *BLAKE2b*, *SipHash-2-4* and *HMAC-SHA256*. The authors concluded that the enhancements made by the usage of those hashing algorithms did not significantly increased the delay on the system, providing an higher level of security.

RaSTA doesn't provide any encryption mechanisms, mainly related with the strict performance requirements, although, one solution that could be applied is the usage of TLS or IPSec, providing message integrity as well as possibly message encryption.

UNISIG SUBSET-098

The Subset-098 maintained by UNISIG, is a specification on how safe communication between Radio Block Centre (RBC) should work on railway signalling systems. UNISIG is a consortium of organizations related to the railways industry, and it was created mainly to the development of technical specifications to ERTMS/ETCS.

This Subset defines a safe communication interface, meaning it is responsible for multiple functions related to safe communications on railways. More precisely, specifies more than safety layers, specifying communication protocols that can be used, having safety layer as part of the standard.

The overall architecture for safe communications is very similar to other protocols, being divided in several modules, each one with a specific safety/security-related function. Are defined two major modules: **Safe Functional Module** and **Communication Functional Module**.

The Safe Functional Module is responsible for providing defences and protection against several threats identified in EN 50159, specifically for Open Transmission Systems (Category 2 and 3). In more detail, it addresses *Repetition*, *Deletion*, *Insertion*, *Re-sequencing*, *Corruption*, *Delay* and *Masquerade* of messages inserted on transmission system. On the standard, it is stated that in order to protect from threats, it implements the security/safety measures recommended on EN 50159 [19], as *Message Authenticity,Message Integrity*, *Message Timeliness* and *Message Sequence*. To achieve this, the module is separated in two sub-modules: *Euroradio SL* and *SAI*.

Euroradio SL is responsible for protecting messages against corruption, masquerade and insertion, providing this by inserting MAC and connection identifier, with source and destination. As for SAI, it is responsible for protection messages against delay, re-sequencing, deletion and repetition, providing this by insertion of delay defence, as EC or TTS and insertion of sequence number. This layer, also provides interfaces for both, application and Euroradio SL, acting as intermediate point. It is specified SAI Protocol on this subset as well as how all mechanisms should be designed.

The Communication Functional Module is responsible for adaptation and transport messages. As main functions, it is responsible for receiving the protected messages by upper layers and managing the redundancy of communication channels. It uses Adaptation Layer (ALE) to manage those channels, specifying how it should be managed, although it is not specified, nether physical links nor the network architecture used. On this layer, is also defined the usage of TCP connections.

In the Figure 3.2 is illustrated the protocol stack of UNISIG Subset-098. There we can see the several layers previously described and how they related with each other. More precisely, we can see the usage of ALE on top of TCP/IP protocols to manage the redundant communication channels and other safety-related layers, as for example EuroRadio Safety Protocol.

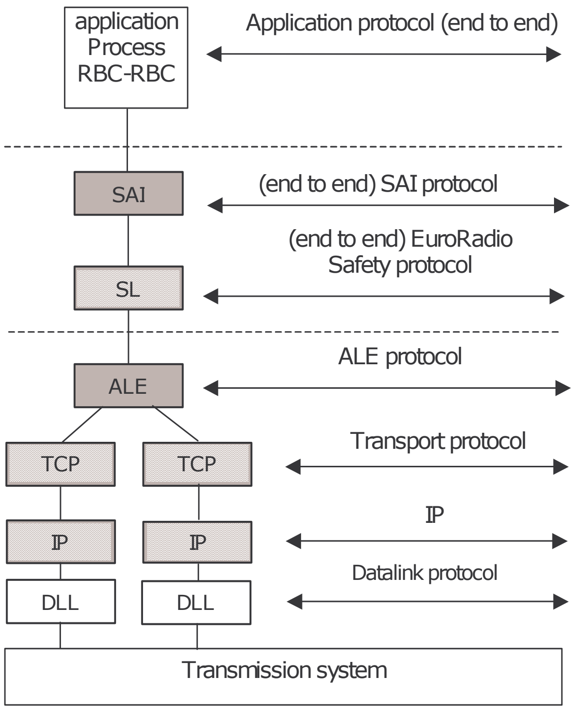

application Process RBC-RBC
Application protocol (end to end)
SAI
(end to end) SAI protocol
(end to end) EuroRadio Safety protocol
SL
ALE protocol
ALE
TCP
TCP
Transport protocol
IP
IP
IP
Datalink protocol
DLL
DLL
Transmission system

Figure 3.2: UNISIG Subset-098 Protocol Stack [56]

For security aspects, Subset-098 itself, addresses partially key management systems and how should it be used, however, this standard doesn't address security directly, leaving it to other Subsets being used, as for example Euroradio. On Euroradio case, we should note the usage of 3DES for the creation of authentication tags to the messages sent over GSM-R. In this case, overall security of the standard could be increased by using a more robust encryption algorithm, such as AES-CMAC, as recommend by [40] and [28].

RSSP-II

Another safety communication protocol is Railway Signal Safety Protocol-II (RSSP-II). This protocol is used in Chinese Train Control System, although not very common on European Union. It specifies a stack of protocols to be used, as well as functional structure for the exchange of safety messages. As mentioned for RaSTA, it is mandatory for these kind of protocols to comply with standards such EN 50159 [19], so this one also addresses both open and closed systems, as defined on that standard.

More precisely, the architecture of RSSP-II is divided in 3 Modules or Layers, the Application, Safe Functional and Communication Functional Modules, each of them taking care of some aspects related to safety and security. The first layer, **Application Layer**, is not directly provided by the protocol stack, but it can be defined by the organization, allowing an easier integration of the safety properties given by the protocol to different devices.

The **Safe Functional Module** is also divided in two sub-layers: *safety application intermediate sub-layer (SAI)* and the *message authentication of the safety layer (MASL)*. SAI has two main functions: adding mechanisms against delay events and sequence numbers (SN) to the messages. RSSP-II provides delay defences recurring to EC or TTS, as they allow to identify old messages. As to Sequence Numbers, they are used with the same objective as in RaSTA, in order to detect insertion, deletion or repetition of a message. The *Message Authentication of safety Layers* is provided by including on the message both identifiers of source and destination, as well as a MAC, in order to prevent message tampering. This layer generates the MAC using the 3DES.

The last layer, Communication Functional Module, is responsible for the transition between safety-related layers, incorporating ALE as the adoption protocol. Usually, this protocol runs over TCP/IP, also allowing a good integration with common and well known protocols and equipment's. Finally, it is stated that RSSP-II has SIL 4.

As mentioned before, this protocol uses 3DES, instead of RaSTA's MD4, to generate the MAC tag appended to messages. This might represent an improvement, although it would still be better to use a more robust algorithm, such as those recommended before. In addition, on this research work, we found lack of information on the way that symmetric keys are managed on the protocol context, what could represent a serious security problem.

3.1.2 Power Distribution Systems

An interesting step to the management of power distribution systems is to move on a distributed self-healing solution, and, in order to do that, we have to consider the sharp increase of the number of nodes on the grid. As mentioned before, nowadays, the network use dedicated channels, what makes it very expensive and complex to use a distributed solution. Because of this, shifting to IP-based networks is becoming very interesting in the area [26] [58]. On the context of 5G Mobilizer project, the usage of public 5G network as transmission system, will simplify the way we connect new devices, however, there are some important points to be taken in consideration.

The distributed self-healing solution on these systems requires that a huge amount of nodes are be able to communicate with each other, as on a peer-to-peer communication. With public 5G network, it is simplified, however, new security concerns arise. If we try to consider the usage of deployed dedicated network on the field, the amount of resources needed to allow this intercommunication would be very high, making this solution unfeasible.

Other important aspect is the network performance. On a distributed self-healing solution, it is required real-time communication between the nodes, with high availability and low latency.

Given the fact that such systems are critical systems, and more precisely, safety-critical, it is important to assure safety. One of the main points where safety might be compromised is by "blinding" some nodes, not allowing the communications to flow. Such situation may happen due to an attack or even by a natural event. So it is very important to enforce safety on the communications. Nowadays are used several protocols to address this problems, where R-GOOSE and DNP3 are two of them, and will be described on the following subsections.

R-GOOSE

Routable-Generic Object Oriented Substation Events (R-GOOSE) is an extension of the Generic Object Oriented Substation Event (GOOSE) protocol which provides fast and reliable mechanisms to provide intercommunication of substations, by means of multicast or broadcast over Ethernet [5]. GOOSE is defined on **IEC 61850** [31]. As an extension of GOOSE, it provides the same main functions but with the possibility of being used over IP based networks.

The main objective of both GOOSE and R-GOOSE is to deliver event data to other nodes on a fast and reliable way. In order to achieve this, the protocol uses multicast and broadcast mechanisms. GOOSE messages are basically data sets of grouped data, as for example in a *(status,value)* format, that are transmitted within a period of 4 milliseconds.

There are various mechanisms applied to achieve the various requirements of the operation environment. First of all, the GOOSE messages are embedded into Ethernet packets, and then a publisher-subscriber scheme is used to deliver the messages. For R-GOOSE, those ethernet packets are encapsulated into layer 3 IP packets. As other mechanism, it implements VLAN tagging, as in **IEEE 802.1Q**, allowing virtual separation and message prioritization. Another requirement addressed was the retransmission of messages. With safety as a priority, retransmission of unreceived messages is a must.

As for the messages itself, they are event driven. As for security, these messages are not authenticated, as for the original standard, although on **IEC 62351** [33], authentication mechanisms are defined to GOOSE messages, however, it is stated that should not be applied if they put in danger the transmission performance requirements. Also, the subscriber do not acknowledge the messages received. GOOSE messages contain a *sequence number* and a *status number.* These fields are meant to prevent replay and insertion/deletion attacks. Finally, to ensure the reception of messages, sent messages are retransmitted until other event is raised.

As illustrated in Figure 3.3 there are several R-GOOSE header fields. From there, we can note the *Security Information* fields, containing data relative to the key used by the security algorithms and the *Signature* fields, holding the HMAC tag of the packet. The *TimeOfCurrentKey* is a 4-bytes long field representing the time in seconds since "epoch". The field *TimeToNextKey* is 2-byte long field containing the number of minutes until a new key is used. The first byte of the *Security Algorithms* field contains the encryption algorithm used, while the second byte contains the algorithm used to generate the MAC. Finally, the *KeyID* field contains the information that identifies the key used, and is set by the Key Management System.

In terms of security, GOOSE mainly addresses data integrity and authentication, on default behavior. This is done by extending PDU with an authentication code, generated by a SHA256 hash using RSA. There is also the need of preinstall the X.509 certificates on system nodes, as certificate distribution is not detailed. Given the performance restrictions, it is not included an encryption algorithm by default, however it is possible to use end-to-end protection as TLS, providing extended security mechanisms. Although, on that case, it is necessary to make sure that proper ciphers are used, in order to not compromise performance of system and time constrains. In [36], Almenares et al. analysed the different ciphers proposed on IEC 61850 [31], and also the usage of more robust algorithms and their impact on performance, concluding that is possible to use stronger algorithms without compromising performance. There are also other possibilities to secure these communications such as the usage of IPSec tunneling, enabling data integrity, authentication and confidentiality.

DNP3

Distributed Network Protocol 3 (DNP3) is a communication protocol stack, used to provide communication between SCADA system components. It is a widely used by US power distribution sector. It is an IEEE-1815 standard, mainly used for substation automation and control systems communication, within energy distribution context, as power and smart grids. This means that it is intended to provided means of communications between stations, remote telemetry units (RTUs) and IEDs.

In a simple way, DNP3 has two main operation modes. There are defined master stations and the outstations. Master stations are control points to the system, while outstations are the ones

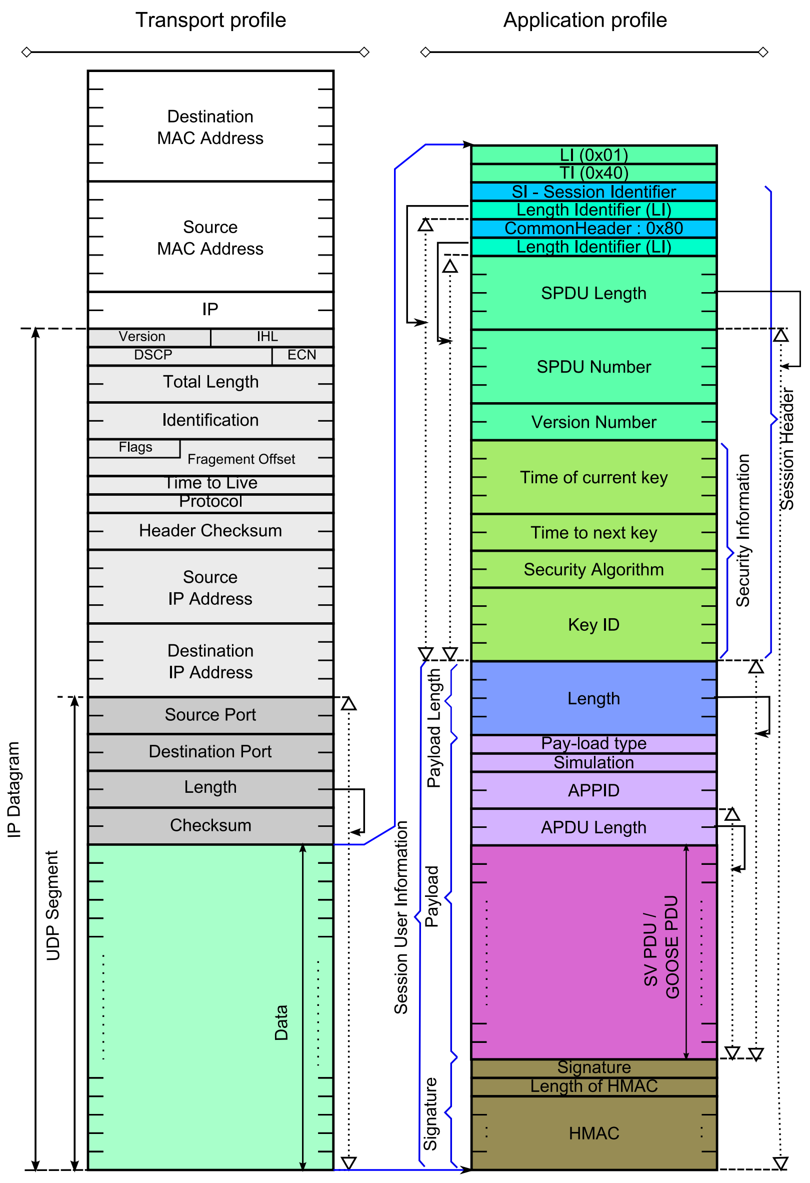


Figure 3.3: R-GOOSE PDU [22]

controlled by master stations. The first operation mode, is when master stations requests data or send commands to outstations, working on a basic Master-Slave architecture. The second operation mode is when an event occurs in a given outstation, and it notifies the master stations without the need of a request. Given this, there are various network architectures that can be implemented: **point-to-point**, **multi-drop** and **hierarchical**. *Point-to-point* architecture is the simplest one, where the system is composed by a master station that can send commands to a specific outstation, as a unicast communication. *Multi-drop* architecture is when a master station can communicate with multiple outstations. In this case, the communication can be done by a multicast or broadcast communication. On a *hierarchical* architecture, one station can be a master station and, at same time, an outstation.

This is a layered protocol, based on Enhanced Performance Architecture (EPA) of **IEC 60870-5** [34]. EPA is composed by **Application Layer**, **Data Link Layer** and **Physical Layer**, although DNP3 added **Pseudo-transport Function Layer** between application and data link layers.

The *Application Layer* is composed by DNP3 user software, providing a set of functions, such as the creation of requests and responses, specifying device identification in terms of being a master or a outstation, providing database storage and also cryptographic security mechanisms. *Pseudo-transport Function Layer* is the intermediate layer between application and data link layers. It is responsible for the first treatment of upper layer messages, mainly dividing data into smaller pieces. The *Data Link Layer* is responsible for various functions, such as retrieving data by polling classes and objects, and other safety/security functions, such as source and destination identification, frame synchronization, flow control, link status, error detection and correction by means of Cyclic Redundancy Code (CRC).

DNP3 is then used on top of a variety of other protocols, such as TCP/UDP and then IP, however it can also be used over serial communications. When used over TCP/UDP - IP, DNP3 data packets are encapsulated on respective protocol packets and transmitted as usual.

As a security enhancement of DNP3, on the application layer, exist DNP3-SA, standing for DNP3-Secure Authentication. This is a security mechanism adopted from **IEC 62351** [33], and its main function is to ensure some security properties on messages exchanged on network. This mechanism generates a MAC tag of the message and appends to it, providing security properties such as origin authentication and data integrity. In order to do this, an HMAC-based algorithm is used, such as SHA-HMAC or AES-GMAC (Advanced Encryption Standard - Galois Message Authentication Code). Inherent to these algorithms, exists an encryption scheme and key management system.

3.1.3 PPDR Platforms

In this subsection, we will present and analyse the communication protocols used within the context of the 5G Mobilizer Project for the BodyKit application, namely those used to enable the communications between the wearable device and the smartphone, the Bluetooth Low Energy Protocol, and the protocol used to enable the communications between the smartphone and the MQTT Server, the MQTT Protocol.

Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a smaller, highly optimized version of common classic Bluetooth technology, being very different on its core specification and design goals. The original main focus of BLE was to provide a radio standard with low power consumption, low cost and low complexity, providing a great data exchange framework, capable of running on several low capabilities devices.

BLE is a protocol stack, being divided in several layers. We can divide the stack on three main groups: Application, Host and Controller. The Application group is the user application that provides an interface to the Bluetooth protocol stack, normally designed with a specific goal. The Host is composed by the upper level layers of protocol stack. The Controller represents the lower level layers of protocol, such as physical layer, the radio.

This technology also have some limitations. Mainly the data throughput can be considered limited, when compared to other wireless communications technologies. This limitation is inherent to the protocol, but it can decrease given the low capabilities of the devices where this protocol is supposed to run. Another possible limitation is the operation range, although, it is not really intended for this technology to have a great range. In theory, this technology is capable of transmitting data up to 30 meters, however, a range between 2 to 5 meters is a more realistic value, given all the adverse possible conditions on real scenarios. Another interesting point of BLE is the two network typologies it allows: Broadcasting and Connections. Broadcasting is when a sending device broadcast data to several observer devices. On the other hand, Connection is when a device establishes a connection directly with other device and then data is exchanged.

This technology uses Secure Simple Paring to connect and paring two devices. This is done by exchanging a 6-digit code, using Elliptic Curve-Diffie-Hellman algorithm. It also uses a shared secret, with the starting point on public key of devices. However, this method has some problems. As shown on [48], it is prone to Man-in-the-Middle attacks, being possible to reveal the shared secret by eavesdropping since the very beginning of paring. On a general view, BLE does not provide user authentication, point-to-point security, or non-repudiation. However, it is possible to use upper layer mechanisms to enhance several security properties, such as encryption, authentication and integrity, but such protocols and algorithms were not designed to run on low-capability devices, meaning it won't be able to work on IoT devices and comply with performance requirements. Finally, there is also other major problem related with this technology, being related with denial of service attacks. There are several attacks that can lead to a denial of service on this stack, compromising the entire systems using it.

MQTT

Message Queuing Telemetry Transport (MQTT) is a protocol for message exchanging that was designed with the objective of being light-weight. This protocol is based on a publisher-subscriber architecture. As mentioned before, it was designed to be used on low-capabilities devices, as well as minimizing the bandwidth usage and computing capabilities of devices, at the sime time that still provides some reliability.

In a more detailed way, MQTT is composed by two types of components: brokers and clients. An MQTT broker is the server that receives messages from several MQTT clients, and that then re-sends it to the respective clients. A client is a device that is connected to the broker and can send and received.

MQTT uses an hierarchical architecture to organize messages and data that receives. This means that, clients subscribe "topics", and publishers send data associated with a given "topic". When a broker receives a message for a given topic, it transmits this data to all client that have that topic subscribed. This allows an high scalability of the system. This also means that, a client device can send data, as for example, sensors data, and at same time receive commands from a CCC. There are various configuration options provided by MQTT and related to the way that topics and message transmission works, allowing several types of notifications. It is also possible to configure redundant brokers, that clients should connect to in case of failure.

The basic MQTT protocol, doesn't provide much security by itself, as it transmits connection credentials in plain text and does not provide any authentication mechanism. Although, is the possibility to use TLS to provide encryption with username and password protection or certificates. However, the majority of IoT devices may not have the capabilities to run such algorithms. Other important point related to security is that, with the publish-subscriber and broker architecture, clients don't know each other IPs, not even what devices/users are connected. There are some research works focused on the usage of lighter ciphers to provide some degree of security on a IoT environment. Among those ciphers are MISTY1 and KASUMI.

3.2 Standards and Requirements

The goal of this section is to present the deep analysis done on several standards that each application must comply with. From this analysis, we collected and defined a set of requirements that each application should meet, so that later we can determine what security properties we need to introduce and the performance restrictions we must not exceed.

Firstly, we will review the most important standards for each application, following a section where we compile that analysis and present our set of requirements.

3.2.1 Railway Signalling Systems

In the world of railway signalling systems, there are several entities that actively produce and disseminate standards, but in the European Union the responsible entity is the European Committee for Electrotechnical Standardization (CENELEC). This committee is responsible not only for the standardization of railway systems, but for general eletrotechnical environment, working together with multiple organizations, in particular with IEC. These standards primarily address security aspects of the system, for both communication networks and for different components of the system.

EN 50159

This standard is titled **"Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems"** [19], and is therefore responsible for indicating the basic requirements to be taken into account in *safety-related* and not *safety-related* communications of the system, but that may interfere with components that are *safety-related*. This is important because in the standard it is stated that if a *safety-related* electronic system exchanges information between various locations, then the transmission system becomes part of the overall *safety-related* system, so communication will need to be secure from end-to-end, complying with this and others standards, notably EN 50129 [20]. Nevertheless, this does not mean that we must apply "all" the security controls and techniques over the network, because, given the environment, there is the need to balance security against performance and safety.

The requirements to which the system must comply can still be divided in two parts, the *safety-related*, which have to be taken into account in the development of the equipment itself, or can be achieved through additional components placed on the transmission system and managed by EN 50129 [20] standard, and the *communication-related* requirements, with the goal of ensuring technical and functional safety, which are addressed in this standard.

The standard begins by stating that it is important to take in consideration the overall architecture of the system, as it can be an open or closed system, which may or may not contain components that are not *safety-related*, or which are not only present, such as a maintenance or diagnostic device. With this in mind, three categories are defined for communications system architecture:

- **Category 1** - Closed system, where the number of components is fixed and known, there is no risk of unauthorized access and the physical characteristics of the system (transmission medium, environment, etc.) are fixed over the life cycle of the system;

- **Category 2** - Open system, however the risk of unauthorized access may be neglected; the number of system components/users is not fixed, and is probably not known, and it may be possible to connect new components, whether *safety-related*, not *safety-related*, or even components that are not related to the system at all; the transmission medium may be of any type and may be influenced by external factors;

- **Category 3** - Fully open system, where unauthorized access is possible; system characteristics as Category 2, however the difference is on the risk of unauthorized access, as on this category cannot be neglected;

This classification is very important to understand which threats the system may be subject to, and the potential risks associated with them. Therefore, this classification is made taking into account three general principles/criteria:

- Knowledge about the network topology, namely the number and type of components, considering whether it is fixed and constantly known;
- Knowledge of the characteristics of the transmission system, including the transmission medium as well as the environment where it is deployed, taking into consideration whether it is fixed and known. As an example, we may think about the usage of copper cables, which will be a known transmission medium and also constant, in contrast to the use of mobile communications medium, which is provided by an external entity (ISP, telecommunications operator, etc.);
- The risk of unauthorised access to the system, whether or not it can be neglected;

The standard identifies several threats that may be present such as broken cables, antenna misalignment, hardware failures, thunderstorms, fires or earthquakes, or intentional attacks to the system, such as wire-tapping, deliberated and unauthorized changes to the system/hardware or unauthorized insertion of system messages. From this set of potentially harmful events, they have defined the following threat categories:

- **Repetition** - Replay of an old message, whether it was deliberated or due to a system component failure;
- **Deletion** - Elimination of a message before it is delivered to the recipient;
- **Insertion** - Insertion of an unauthorized message in the system;
- **Re-sequencing** - Changing the sequence number of a message; This could be caused by a system error (hardware failure), or by an external agent, causing the message to be considered invalid;
- **Corruption** - Changing some bits during message transmission, causing message corruption;
- **Delay** - Delay in message delivery; It can be caused by normal network congestion (not being an attack), or due to and external agent (attack);
- **Masquerade** - When an attacker impersonates a legitimate component of the system and directly interfere with it, leading to potentially unsafe situations;

Table 3.1 shows the relationship between the system categories and the threats associated.

To mitigate those threats, a set of defenses are suggested, in which their application is directly related to the system category:

- **Sequence number**
- **Time stamp**

Table 3.1: Relationship between system categories and threats [19]

Category	Repetition	Deletion	Insertion	Re-sequence	Corruption	Delay	Masquerade
Cat. 1	+	+	+	+	++	+	-
Cat. 2	++	++	++	+	++	++	-
Cat. 3	++	++	++	++	++	++	++

- Threat can be neglected.
+ Threat exists, but rare, weak countermeasures sufficient.
++ Threat exists; strong countermeasures required.

- **Time-out**
- **Source and destination identifiers**
- **Feedback message**
- **Identification procedure**
- **Safety code**
- **Cryptographic Techniques**

Finally, it should be noted that among the defenses presented, the standard points out that if there is a possibility of unauthorized access, such as in the case of Category 3 systems, additional security measures must be added, as **conventional security techniques** and applying **specific cryptographic techniques**, for each specific case.

In the context of 5G Mobilizer Project, we are shifting the railway signalling system from a closed or semi-closed network, to a fully-open network, therefore, we can categorize it as a Category 3 System. Consequently, we must consider that all of the threats before mentioned exists. Firstly, we have to consider general security requirements of safety-critical systems, namely the high availability of the system. This is one of the most important requirements because if, for any reason, the system is unavailable or it is not capable of receiving or sending information, it will drastically increase the risk of occurring an accident. More related to railway signalling, in a Category 3 system, unauthorized access must be considered meaning that message authenticity is a must. Allied to message authenticity we have to consider message integrity, in order to prevent message corruption that could lead to system manipulation. Common to other critical systems, message timeliness is also extremely important, as we need to have low latency and low response time for message exchange. Finally, in this specific context, it is also important to ensure the messages sequence. An "out of order" message, could lead to the system on entering in a erroneous state, as the information inside that message may already be different from the reality, even if that message came within a valid time frame.

On the Table 3.2 we present a summarized version of the requirements for railway signalling system, focusing on the communication channel.

3.2.2 Power Distribution Systems

In terms of standards, there are several related to the system's design as well as for protocol stacks to be implemented, as for example **IEC 61850** [31], that describes a protocol stack to be used for communications on power distribution systems, with R-GOOSE being one of them. Related to security, exists one major standard, **IEC 62351** [33], that specifies several mechanisms

Table 3.2: Railway Signalling System requirements and their description

Requirement	Description
System Availability	The system must be available any time it is required, even if a given communication line is down
Message authenticity	Must be possible to identify and confirm who is the message sender
Message integrity	Must be possible to verify if the message received did not suffered any change since it was sent (either by natural or unnatural causes)
Message timeliness	The messages should not be accepted if are not "fresh" but also, we should guarantee the low latency of communications
Message sequence	Messages older than the last accepted message should not be accepted

that should be added on top of other standards, as for example **IEC 61850**, and protocols, such as R-GOOSE. On the next subsection will be described IEC 62351, specially parts related with protocols that are going to be used, such as R-GOOSE, and mechanisms related with network and communications.

The development of these standards is done by International Electrotechnical Commission (IEC), that created a technical committee, **IEC TC57**, with the responsibility of creating standards for data exchange and transmission in energy and other systems where this kind of guides can also be applied, such as SCADA and distributed automation systems.

IEC 62351

This standard was developed to address specifically security issues on systems and protocols used on power distribution environment, mainly those developed by *TC57*. It specifically addresses security related on the data exchanged between several components of the system, trying to achieve security properties such as confidentiality, integrity, non-repudiation and availability. A relevant aspect of the standard is that it gives specific technical details on the security mechanisms recommended.

The standard is divided in ten parts, each one addressing a specific part of the overall system, as for example protocols based on TCP or protocols described on IEC 61850, as R-GOOSE. On this section, the standard will be described, pointing out the most relevant parts to this research work.

The first mechanisms provided are for TCP/IP based protocols. For this protocols, the standard recommends the usage of Transport Layer Security (TLS) with X.509 certificates. This provides two main properties: *authenticity* and *integrity* of data, preventing data tempering as well as impersonating or message insertion, mainly mitigating man-in-the-middle attacks. Other property that can be ensure is *confidentiality.* This mechanism intents to provide secure communications from point-to-point, and device authentication. From a technical perspective, the standard gives support to revocation and validity analysis of certificates. More related to ciphers used, it does not states what to use, although, it requires the support of RSA and DSS for digital signature algorithms, as well as specifying the usage of 2048 bit keys with RSA. For the key-exchange, it requires support to Diffie-Hellman, with 2048 bit key, however, 1024 keys could also be supported for backwards compatibility.

The standard also addresses security for 60870-5 and similar standards and protocols, such as DNP3. For these, it is recommended the usage of a challenge-response authentication mechanism, using HMAC with pre-shared keys. This mechanism aims to ensure data integrity.

Other protocol inside the scope of this standard is R-GOOSE and IEC 61850, namely Sampled Values (SV). In this case, there are usually strict requirements related to the performance of the system, what makes difficult to implement good security mechanisms. When it is possible, it is recommended to add an RSA-based signature to the PDU, ensuring data integrity of those frames. Also, if possible without compromising performance requirements, could be applied encryption mechanisms.

The last important part is related with access control to the system, stating that is important to specify a role-based access control, with minimum privilege principle and pre-defined rights.

There are some analysis already done on this standard, such as [44], where Schlegel et al. point out some restrictions on the standard. As for the TCP/IP based protocols, it does not specify the cipher used by TLS, opening the possibility of usage of insecure algorithms, such as RC4. Also, the standard allows the usage of TLS without encryption, what can always be seen as a downside. For IEC 60870-5 and alternatives such DNP3, it does not addresses encryption, not providing confidentiality to the communication. Also, [44] points out a potential problem of this mechanism, where an attacker can make the system perform denying actions, such as session keys invalidation, allowing a Denial-Of-Service to be successful. Finally, for the R-GOOSE and IEC 61850, the recommendations might be very difficult to implement without affecting performance. Usage of RSA won't be possible on application with 3ms response time restrictions, as demonstrated by [21] and [24]. Although [44] proposes the usage of HMAC, that can be implemented in hardware, or even HMAC implemented in software, given that it computational overhead is significantly minor and complies with time constrictions.

IEC 61850

The analysis of the performance requirements of this system plays an important role on this work, as adding extra security mechanisms will certainly add delay to the communications, being necessary to assess the time window that exists. These requirements are established on the fifth part of IEC 61850 standard [31], addressing specially the performance requirements for R-GOOSE messages.

The standard specifies a maximum message transfer times, given the type of message and the application. The transfer time is the complete transmission of a message, including time that devices take to handle it at both ends, meaning it starts when the sender puts some data on the transmission stack, until the moment the receiver retrieves it from its transmission stack. Figure 3.4 illustrates transfer time.

Given this, transfer time includes processing time in both physical devices (t_a and t_c) and transmission time over the transmission system (t_b).

There isn't a single transfer time requirement for transmitting messages, as different applications and functions inside the system might have different necessities. Messages are divided in types, where they are grouped by similar performance needs. Some message types are also divided in performance classes, as, depending on the context of the event, it may require different timings. There are the following message types defined:

- **Type 1A** - Fast Messages - Trip - Example: Circuit breaker commands, States
- **Type 1B** - Fast Messages - Others - Example: Circuit breaker commands, States

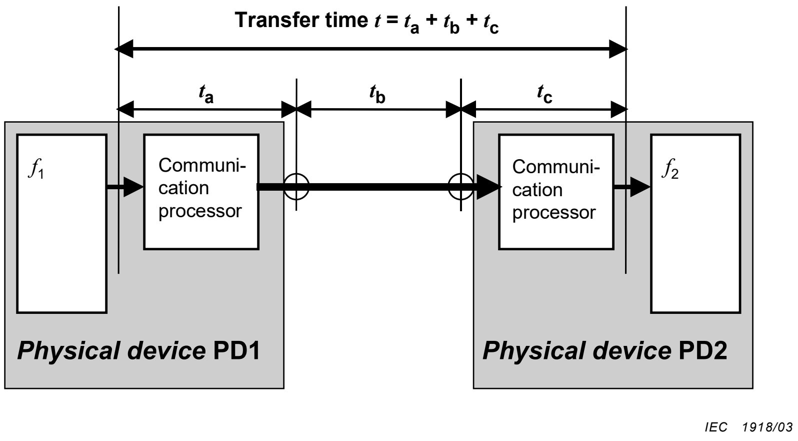


Figure 3.4: Transfer time representation [31]

- **Type 2** - Medium Speed Messages - Example: RMS values from Raw Data Messages
- **Type 3** - Low Speed Messages - Example: Configurations, Non-electrical measures
- **Type 4** - Raw Data Messages - Example: Digital electrical measurements
- **Type 5** - File Transfer Functions - Example: Data for recording
- **Type 6** - Time Synchronization Messages - Example: IED internal clock synchronization
- **Type 7** - Command Messages with Access Control - Example: Messages with Secure Procedure

On the Table 3.3 are described the transfer time requirements for each message type and performance class.

Table 3.3: Table of Maximum Transfer time for each Message Type and Performance Class [31]

Type	P Class	Max. Transfer time	Application
1A	P1	10ms	Fast M. - Trip
	P2/3	3ms	
1B	P1	100ms	Fast M. - Others
	P2/3	20ms	
2	-	100ms	Medium S. M.
3	-	500ms	Low S. M.
4	P1	10ms	Raw Data M.
	P2/3	3ms	
5	-	≥1000ms	File Transfer
6	-	-	Time Sync.
7	-	-	Control Commands

For the scope of this work, we will focus on both **Type 1A** and **Type 1B** messages, as they are the ones transmitted by R-GOOSE protocol.

The same way as in the railway signalling systems, in the context of 5G Mobilizer Project, we are moving to fully or semi-open networks, therefore, several new security threats arise. Again, as a critical system, we have to consider availability as a main requirement for the system and for the communications. From the standard IEC 62351 [33], we understand that message authentication and message integrity is also a must. Obviously, as we may face unauthorized access we must ensure that only valid messages, that were sent by authorized devices will be accepted, as well as only messages that were not tampered are accepted. In the same standard, we can see that confidentiality should also be considered, specially if the system uses open networks, although, this should not compromise the timeliness and low latency/response time of the system. Finally, from the standard IEC 61850 [31] we can specify the exact values that this system has to comply with in terms of latency. The values, as shown on the Table 3.3, are between 3ms and 10ms, meaning that very low latency is required on the communications. This requirement is of particular importance for testing and validation of our solution, as performance can not be decreased to a latency higher than 3ms.

On the Table 3.4 we present a summarized version of the requirements for power distribution systems, focusing on the communication channel.

Table 3.4: Power Distribution System requirements and their description

Requirement	**Description**
System Availability	The system must be available any time it is required, even if a given communication line is down
Message authenticity	Must be possible to identify and confirm who is the message sender
Message integrity	Must be possible to verify if the message received did not suffered any change since it was sent (either by natural or unnatural causes)
Message confidentiality	Must be impossible to an unauthorized user understand the contents of the message, the messages should be encrypted
Message timeliness/very low latency	The messages should be delivered in the minimum time possible, never exceeding the maximum Transfer Time of 3ms/10ms (depending on the scenario)

3.2.3 PPDR Platforms

The analysis of the requirements for PPDR Platforms were not made based on standards, instead was done based on several documents related to the 5G Mobilizer Project, specially the document "Deliverable D3.2 and D3.3" [47], containing the security requirements for this system, and the analysis of the system and it's communications.

Again, as a critical system, one of the major requirements is availability. If, for any reason, a part of the system or even the communication channel became unavailable, the entire mission and the lives of the assets may be endangered. This system will use two types of communications: BLE and 5G networks. In both of them, the communications are prone to man-in-the-middle attacks, leading us to the message integrity and message authentication requirements. At the same time, message confidentiality could also be included as open-networks are used, although this should not be a priority as the devices used have low capacities and should be considered lightweight methods. Finally, another requirement is mentioned on [47], privacy, although not having maximum priority. This system directly deals and exchanges data collected from sensors belonging to operations on field, therefore data privacy should be preserved, but only on scenarios where it does not penalize significantly the overall performance of the system.

On the Table 3.5 we present a summarized version of the requirements for PPDR platforms, focusing on the communication channel.

Table 3.5: PPDR Platform requirements and their description

Requirement	**Description**
System Availability	The system must be available any time it is required, even if a given communication line is down
Message authenticity	Must be possible to identify and confirm who is the message sender
Message integrity	Must be possible to verify if the message received did not suffered any change since it was sent (either by natural or unnatural causes)
Message confidentiality	Must be impossible to an unauthorized user understand the contents of the message, the messages should be encrypted
Data privacy	The data inside each message should not be associate with the individual who generated it

3.3 Gap Analysis

In this section, we will use the analysis done on the previous two sections to determine the gap between what protocols that will be used already provide, in terms of security, and what will be needed, given the shift to 5G technology. On the context of 5G Mobilizer Project, the communication protocols are already defined, being *RaSTA* for the railway signalling system, *R-GOOSE* for power distribution system, and *BLE* and *MQTT* for the BodyKit Platform. For the power distribution system, the already deployed mechanisms taken into account were based on the practical implementation we had access, and from the knowledge shared by other 5G Mobilizer Project partners, more precisely, EFACEC.

The main difference we can see from the usage of 5G networks on the specified scenarios, is the shift from a closed and isolated network, to an open network, being prone to several new types of attacks, namely those related with unauthorized access and eavesdropping. It is also important to note the different security requirements of each application, where, for example, confidentiality is less relevant than message integrity or system availability. For a proper gap analysis we need to have in consideration the security requirements that these scenarios demand, already having in consideration the usage of 5G networks, and the security mechanisms already in place.

In a general overview, the system availability is a requirement common to all of the critical systems and it is already addressed in all of our scenarios. RaSTA, R-GOOSE and PPDR plataform communication protocols, already implement mechanisms to enforce high availability, as for example, transmission channel redundancy and message acknowledgements.

For the railway signalling system, we summarize the mechanisms that RaSTA already provides and the requirements each one addresses in Table 3.6. From the table, we can see that all of the requirements have at least one mechanisms deployed, although, RaSTA and these mechanisms, were developed for closed networks and, when deployed on open networks, these mechanisms don't guarantee the compliance with the requirements. More precisely, the message authenticity is ensured on closed networks by source and destination identifiers inserted on the packet, as well as the procedure identification. However, if an attacker captures a packet, he can easily craft valid source and destinations [28]. As for the message integrity, the Safety Code is an MD4 hash code of the entire PDU, providing integrity protection on close networks, but on open networks an attacker can easily tamper with the packet and update the safety code. The same could be applied to message sequence, as sequence number of the message can easily be changed to perform several types of attack, invalidating valid messages, for example [28].

Table 3.6: Relationship between deployed RaSTA mechanisms and security requirements

Requirement	**Mechanisms**	**Effective on Open Networks**
System Availability	Redundant Communication Channels	Yes
Message authenticity	Source and destination identifiers Identification procedure	No
Message integrity	Safety Code	No
Message timeliness	Timestamp Time-out	Yes
Message sequence	Sequence Number	No

On the power distribution systems, we will make this analysis comparing the mechanisms that are common nowadays on the industry (knowledge shared by EFACEC), with the requirements elaborated before. Table 3.7 summarizes and relate the deployed mechanisms with the security requirements. Although in several standards related with R-GOOSE there are recommendations to add security measures, namely for data authentication and integrity, a vast majority of the implementations don't support that. Those standards, as IEC 62351 [33], recommend the usage of asymmetric cryptography (RSA), which compromises the very low latency requirement. In terms of confidentiality, the problem is similar, as the standard also proposes the usage of encryption mechanisms including asymmetric cryptography, although it will severely compromise the system's performance.

Table 3.7: Relationship between deployed R-GOOSE mechanisms and security requirements

Requirement	**Mechanisms**	**Effective on Open Networks**
System Availability	Redundant Communication Channels Message Retransmission	Yes
Message authenticity	None	No
Message integrity	None	No
Message confidentiality	None	No

On the context of PPDR platforms and more precisely BodyKit, we analysed the gap only for the protocol that will be used over 5G, that is MQTT. From the document [47] and the protocol analysis, we can verify that MQTT already allows the integration of SSL/TLS on its communications, although it might be heavy for the devices used. Considering this possibility, we can state that this protocol already addresses message authentication, message integrity, with the possibility of achieving message confidentiality. As for data privacy, MQTT does not includes any mechanisms to ensure such property. Table 3.8 summarizes the requirements and mechanisms already deployed for PPDR platforms.

The Table 3.9 presents the requirements for the applications, including it's priority accordingly to the analysis previously done, and identify which requirements are already addressed by each protocol. From the analysis of Table 3.9,we can see there is a set of common requirements among the three systems. It is important to note that the top priority requirements are also clearly identi-

Table 3.8: Relationship between PPDR Platform over-5G protocols (MQTT) deployed mechanisms and security requirements

Requirement	**Mechanisms**	**Open Network effective**
System Availability	Redundant Communication Channels Message Retransmission	Yes
Message authenticity	SSL/TLS	Yes
Message integrity	SSL/TLS	Yes
Message confidentiality	SSL/TLS	Yes
Data Privacy	None	No

Table 3.9: Application's requirements and current state

Application / Requirement	Railway Signalling	Power Distribution	PPDR Platform
System Availability	🟩 5	🟩 5	🟩 5
Messages Integrity	🟨 5	🟥 5	🟩 5
Messages Authentication	🟥 5	🟥 5	🟩 5
Messages Confidentiality	🟥 4	🟥 4	🟨 4
Messages Sequence	🟩 5	⬜ -	⬜ -
Data Privacy	🟥 1	🟥 1	🟥 3

🟩	Already achieved
🟨	Not totally achieved
🟥	No controls deployed
⬜	Not applicable

1 - Lowest priority ; 5 - Top priority

fied, being **System Availability**, **Message Integrity** and **Message Authentication**. Given the very low latency requirement and the heavy computational capacity that encryption requires, including encryption on this systems may not me possible, although, we consider it as an important requirement when shifting to open networks. Lastly, Data Privacy can be neglected in both railway signalling and power distribution systems, given the type of data that RaSTA and R-GOOSE messages exchange. However, in PPDR platforms, it should be considered as relevant, as it exchanges data directly related with the field operationals. The "color classification" of the cells, reflects the implementation or not of security mechanisms that enforce such requirements, allowing us to understand what is needed in each context. Our objective is to achieve the top priority requirement that are not met yet, being **Message Integrity** and **Message Authentication**, followed by **Message Confidentiality**.

3.4 Security in Critical Systems Communications

There are several work already done on this areas that are important mention and to have in consideration while developing this research work. Firstly, we will review work done on securing railway signalling systems, related to analysis and enhancements proposed to specific protocols, such as RaSTA, or system architectures/solutions that aim to improve the overall security of the system. Then, will focus on security-related work on the context of Power Distribution systems, namely R-GOOSE and its security, GOOSE protocol and several security analysis that identified multiple vulnerabilities, research assessing the overall security of IEC 61850 [31] standard, which GOOSE is part of, and several proposals to enhance security in these topics.

Security in Railway Signalling Systems

On the railway signalling security context, some projects and research work were conducted. Firstly, in 2003, Smith et al. [52] introduced security as a problem capable of generating safety issues, raising the awareness to the need of implement security mechanisms on these systems. They analysed railway systems, mainly railway signalling system, identifying several vulnerabilities and proposed a set of countermeasures, aiming to preserve Authentication, Confidentiality and Integrity of the system. Those proposes were made having in consideration the developments of that time, mainly by means of cryptography functions. They also used a real project as "testbed" for their proposals, namely the Australian Rail Network.

In 2015, American Public Transportation Association (APTA) published a white paper on *Attack Modelling* [4] on the context of Securing Control and Communications Systems in Rail Transit Environments, followed by several other recommendations. As outcome of the work, they identified as major properties to have in consideration the integrity and availability of the system. They also specified that asset characterization should be done based on the importance it has from a safety-critical perspective of the system.

Other project conducted on this topic is Cybersecurity for Critical Infrastructures (CYSYS). CYSYS is a German group composed by industrial and academic entities, which collaborate to identify, investigate and resolve problems related to critical infrastructures, with railway infrastructure as one of them. The project is still on going but some proposals already were made, as the introduction of the concept of the security shell to the interlocking and signalling systems [23].

Previously, in 2012, started the project SECRET (SECurity of Railways against Electromagnetic aTtacks). Although this research work isn't directly related to the focus of our work, it is interesting to mention as another project that addressed the information and computer security on the railway signalling context. This work [55] analysed attacks and vulnerabilities that could be exploited by electromagnetic interference to the European Rail Traffic Management System (ERTMS), raising the awareness to cybersecurity on the area.

Another project related to SECRET is ARGUS [35]. ARGUS was focused on railway signalling systems. The main objective was to define best practices on how to apply security to railway signalling systems. They stated that in order to design a good security solution is necessary to take in consideration security of the network, deployment of security mechanisms and also the signalling security itself. They also propose risk analysis and assessment models to be applied in this area.

In [49], Vateva-Gurova et al. analyse the challenges of including security in a safety-critical environment like railway signalling systems. They state that one important think to have in consideration is the several standards and regulation that software and hardware included on these systems must comply with, and that changes made to it must also be re-validated under those standards. At same time, the life-cycle of a security component isn't always linear and changes must be done very quickly. That was one major problem they identified, because if such security components are embedded on safety-critical components, any change must be re-validated, not allowing dynamic modifications. With this in mind, they propose the usage of a shell concept to address this problem. More precisely, they propose the development of a "security box" to be placed between the several system components that communicate over the network and the network itself. This box should provide security functions like integrity and confidentiality. With this approach, it is possible to separate security from safety at some level, solving the first problem, at the same time as it is possible to integrate several security functions to ensure different security properties, while establishing a defence-in-depth approach.

In [10] and [27] M. Heinrich et al. kept developing the work done on [49], enunciating security requirements to safety-critical railway signalling networks. They considered the entire system, designing a solution to enforce security not only on the transmission system but to the entire system. They stated specific requirements, such as confidentiality of classified data to illegitimate users, or not usage of insecure transfer methods, but also included standard-related requirements, such as the ones from EN 50159 [19]. They also defined the architecture to the solution they propose. As previously, they propose the usage of shell concept, introducing a "security box" to the system. To do this, it is purposed a Multiple Independent Levels of Security (MILS) platform, that consists of a separation kernel, the underlying hardware and several security partitions. They propose the usage of a Trusted Platform Module 2.0 (TPM) to provide secure storage, execution of cryptographic functions and other features, such as authenticated boot and update procedures. On top of that, several security functions can be added, being proposed Intrusion Detection Systems, Firewall, Remote Attestation, Health Monitor, etc. As it is a on going project, work is still in progress and by this time, there are not publicly available experimental results of the proposals.

At protocol level, there was some work done on RaSTA security analysis and some enhancements proposed. In [28], M. Heinrich et al. analysed RaSTA protocol from a security point of view, focusing the cryptographic functions and concepts it uses. On this paper, they analyse RaSTA protocol and identify its major weaknesses. They concluded that the cryptographic algorithm used to generate the 8B checksum (*safety-code*), an MD4-based hash function, is prone to several attacks and is considered a weak cipher. Moreover, they point out that the network key used as shared secret for authentication, can be disclosed or, if known by an attacker, he can forge valid messages without much effort. As proposal, they made some enhancements to the protocol, namely by replacing MD4 algorithm for a more robust option like SipHash-2-4, BLAKE2 or HMAC-SHA256.

They validated their choices through experimental work, analysing the impact of the enhancements on performance. MD4 was faster then all the new enhancements, as expected, however they stated that every enhancement could be applied without compromising system performance. More precisely, a single machine must be able to calculate 2000 RaSTA instances, meaning it has to generate 6667 (2000 instances * $\frac{10}{3}$ (an heartbeat at every 300ms)) safety codes per second, plus being able to calculate other 6667 instances to validate those codes. From their experiments, SipHash could perform 1 388 596 calculations per second, BLAKE2 perform 1 089 137 and even HMAC-SHA256 could produce 499 676 code, meaning that those 13 334 hash calculations are just a fraction of the throughput this algorithms can provide. This numbers are relative, as it varies with the computing resources used, although the authors recommended SipHash to replace MD4, and stated they believe that even HMAC-SHA256 could be used without compromising performance requirements (Figure 3.5).

As final enhancement, they propose the usage of asymmetric cryptography (i.e., digital signatures) to solve the network key problem, although the main drawback is its performance. Finally, they point out that RaSTA packets could be sent through a secure transport channel, using IPSec or TLS, however they don't provide any experimental results to show it is a feasible solution in terms of performance.

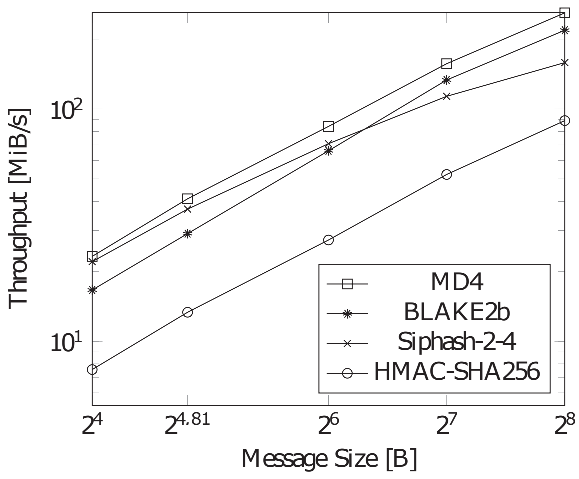


Figure 3.5: Throughput comparison of proposed algorithms by M. Heinrich et al. [28]

Security in Power Distribution Systems

There is a lot of research done on the context of security in Power Distribution Systems, with several focus. It is relevant to analyse topics such GOOSE security analysis and enhancements because R-GOOSE is very similar to it [5], giving an insight of possible security problems that exist in both GOOSE and R-GOOSE.

Several authors focused their research work on GOOSE protocol, analysing the security properties it provides and identifying vulnerabilities, designing attacks and suggesting countermeasures. Firstly, in [29] and [39], GOOSE protocol is analysed in terms of possible vulnerabilities. N. Kush et al., in [39], looked to how GOOSE receivers (subscribers) processed the messages, identifying a severe vulnerability through the ***StNum*** (Status Number) field. The vulnerability consisted on the message validation. When a message is received, the subscriber checks the StNum and compares it with the StNum of the last message accepted. If the new message contains a lower StNum, then it is an invalid message, unless a roll-over occurs. If an attacker could forge GOOSE messages with extremely high StNum, then it would invalidate all true-valid messages. The authors designed three attacks. The first one was very basic, the attacker just send one packet with StNum equal to $2^{32} - 1$, being the maximum number without triggering a roll-over. The second attack was a High Rate Flooding Attack, where the attacker keep sending GOOSE messages with increasing StNum, in a high rate. This means that, at some point, even if the attacker started with a lower StNum than the real messages flow, the attack flow StNum will become higher, invalidating legitimate messages. The third attack was an "intelligent" flood attack. The attacker first analyses normal traffic and determines current StNum, as well as the normal messages rate. Then the flood attack begins with a slightly higher rate and already with higher StNum, being a more stealthy attack. The attacks were all tested and successful.

In [29] and [13], J. Hoyos et al. and M. Franco, respectively, explorer a vulnerability on the GOOSE message PDU, as they where able perform a spoofing attack, intercepting a message, changing the reading boolean field and re sending it to the network, being accepted by the subscriber. In [13] author proposed the usage of VLAN Tagging and SDN techniques to mitigate such vulnerabilities, mainly by specifying exactly the traffic allowed on the SDN network.

In [51], Dae-Yong Shin et al. performed a security analysis of SCADA systems based on IEC 61850 [31]. They considered the need for a protocol that guarantees message authentication and integrity in such systems, having in consideration message-based communications protocols present in the standard, such as GOOSE. Was identified the lack of mechanisms to enforce such properties and with that in mind they proposed the usage of message authentication codes, with several variants, namely the usage of Block Ciphers and Keyed-Hashing. On the first method, they proposed to encrypt a message in cipher block chaining (CBC) mode, and only keep the last block, to be used as MAC for the message. They considered the usage of AES as block cipher. To the second method, they analysed the usage of a HMAC-MD5 based MAC. Both solutions require a key sharing mechanism, which they also have in consideration, using a trusted server of SCADA system to manage such keys.

Later, in 2016 and then in 2019, several works [44] [30] [11] proposed enhancements directly to the GOOSE protocol and analysed recommendations from IEC 62351 [33]. In [44], R. Schlegel et al. analysed the recommendations made on IEC 62351, including one part relative to GOOSE messages. They suggest the usage of the PDU Extension, being used a signed hash to authenticate the messages. It is suggested the usage of RSA-based signatures, however the authors find it controversial because of the increased latency that asymmetric cryptography will add to the system. Given that, the suggest the replacement for a HMAC-based or even Elliptic Curves cryptography.

In [30], S. Hussain et al, analysed the usage of Message Authentication Codes for securing GOOSE messages. On their work, they choose several algorithms and implemented various versions of a modified GOOSE protocol, S-GOOSE, to include such MACs. They choosed AES-GMAC and HMAC-SHA256 as algorithms to test and, on a laboratory environment, they were able to conclude that both solutions, with a pre-shared key, performed in approximately 0.1ms, not compromising the 3ms of maximum latency.

Later in 2019, K. Boakye-Boateng and A. Lashkari [11], presented a solution where a One-Time Pad scheme could be used to provide encryption and also authenticity and integrity to GOOSE messages. The idea was to encrypt the important fields of the message with OTP algorithm. More precisely, it would generate a random, same size of message, byte array that would be used as key, perform a XOR operation between the message and that key, and send the result byte array. They tested the algorithm with some success, even if that solution require some more research, specially on the PRNG that should be used, given the capabilities of the devices.

There were some work developed focusing on analyzing the standard IEC 62351 [33], which is responsible for presenting security recommendations on several other standards like IEC 61850 [31] and therefore, GOOSE. In 2010, S. Fuloria et al [25], analysed Part-6 of the standard, focusing on mechanisms to enforce authentication and integrity in GOOSE messages. The recommendation is to include RSA-based signatures on GOOSE's PDU. They implemented such mechanisms and tested if it was a feasible solution. They concluded that even hardware-based solution of such algorithms would take more than 4ms, compromising the existing performance requirements. In order to overcome such problem, they suggest the usage pf Elliptic Cryptography as it could achieve an interesting performance (around 1ms).

In a broader perspective, some authors analysed the security that is already in place on IEC 61850 based systems, having in consideration the general recommendations. On that field, [46] and [18] analysed how these systems are designed, the security requirements it needs and also the existing mechanisms. In [46], U. Premaratne et al, identify the main threats to these systems like system disruption and gaining access to confidential information, proposing several attack scenarios and mitigation strategies. To validate their propositions, they did an extended security assessment on a testbed, identifying multiple attack vectors. They conclude proposing the usage of IDS, network and host-based, as a viable solution. In [18], A. Elgargouri et al, analyse security requirements of these systems, mainly focusing on the attack detection and self-healing of the grid. They don't propose specific enhancements to protocols but, similar to [46], they recommend the usage of IDS to detect such attacks. In [37], the authors propose and implement a security gateway to provide security properties on power distribution systems. In this case, they aim to protect synchrophasors data, providing such properties to the protocol R-Sampled Values.

Finally, there are already some analysing and enhancements proposed to R-GOOSE protocol. In 2016, J. Schuler et al [50], analysed if it was possible to use Digital Signatures on R-GOOSE messages and what were the best algorithms to perform such task. They compared Salsa20 [9], Poly1305 [8], AES, SHA1, SHA256 and HMAC. After an extensive practical analysis, they concluded that Salsa20 and Poly1305 allowed to sign R-GOOSE messages in around 14us. They also performed a more complete analysis on these algorithms, calculating the complete transfer time to be around 668us, meaning they are feasible algorithms. Other work focusing on R-GOOSE was [12], where D. Saraiva et al. compared the usage of GOOSE and R-GOOSE in the same LAN and over a WAN. They also analysed the usage of IP or VPN and concluded hat both accomplished the performance requirements mandatory by IEC 61850 [31]. Their work was performed with real mobile and landline networks, differenciating from all of the other works based on simulators.

3.5 State of the Art Conclusions

On this chapter we analysed the most important communication protocols used on the three critical systems on study in this work and we performed a critical analysis on the security they already provide. From that analysis, we could state that these system were designed without having security first, mainly because safety comes first, meaning it is crucial to achieve high performance. On the other hand, security on the communication channel was relegated to second place given the fact that these systems were designed to operate on top of a closed network, with very low probability of occurrence of an attack.

Secondly, we reviewed the main standards and regulation from each area related with security and performance. From those standards and the review we did on the communication protocols, we performed a gap analysis between the requirements that each system would require to operate in a open network, and the security properties that the current protocols already provide. From that analysis, we realized that we need to address the following requirements in terms of message security:

- **Message Integrity**
- **Message Authentication**
- **Message Confidentiality**
- **Very Low Latency**

Although, we also stress the necessity of ensuring both **High Availability** and **Very Low Latency** on our solution with maximum priority.

Finally, we reviewed some work already done in these areas. On railway signalling systems, we identified two major solutions to enhance security, changing the system architecture to include security by design, or adding security mechanisms to specific protocols, such as RaSTA. In [49] [10] [27] the authors proposed a shell concept, where the core protocols and application could be "inside" a security module that would provide security functions. In [28], the authors proposed changing the outdated MD4 algorithm used in RaSTA to generate the *safety-code*, for a more robust option like SipHash-2-4 or BLAKE2. On power distribution systems, several work were conducted to demonstrate practical attacks and identify vulnerabilities on actual systems [39] [29] [13], with emphasis on GOOSE protocol. There are also several proposals to enhance security on GOOSE, by adding extra fields to it's PDU, as for example, an hash tag to validate integrity and authentication of the packet [44] [30], or packet encryption [51] [11]. Other works focused on evaluating performance of other security techniques, such as IDSs [46] [18]. Lastly, one research work [50] focused on R-GOOSE, and analysed the usage of Digital Signatures to provide authentication and integrity. They proposed the usage of Salsa20 or Poly1305.

Chapter 4

Proposed Solution and Implementation

In this section we will present the initial experiments where we analysed the available open-source implementations for the applications of 5G Mobilizer project and we describe the modifications made to such implementations. Then, we propose a solution to provide the security requirements defined on the previous chapter, describing the decisions taken during the development process and discussing the implementation details. Furthermore, we explain the several scenarios on how our solution could be applied, using the integration with 5G Mobilizer project as a practical example.

Due to the lack of open-source implementations of communications protocols from the scenarios of 5G Mobilizer project, that is explained on subsection 4.1 above, this proposal is focused on the Power Distribution systems scenario. Our goal is to provide security mechanisms in a scenario that demands very low latency.

4.1 Initial Experiments

In this subsection, we will present the initial experiments performed on the contexts of the 5G Mobilizer project applications. The goal of these experiments were to evaluate the available open-source implementations of protocols and devices that could be used to implement and validate our proposal. Based on these experiments we performed a set of modifications to the implementations that we will explain in the next subsections.

4.1.1 Implementations

In this subsection will be addressed the several solutions that can be explored during this research work, related to the protocol implementations, operating systems and devices used on the context of each PPS3 application. Our main focus is to develop security mechanisms that could be applied to all of PPS3 applications. However, at the experimental level, it was extremely difficult to find any available implementation or environment for Railway Signalling and PPDR systems. Given that we decided that our experimental work will be focused on Power Distribution Systems.

Thus, there are several parts important to analyse on the testing environment for the lab work. First, it is important to emulate the normal traffic that exists on such systems, mainly, the R-GOOSE messages. To achieve this, will be needed a library or an open implementation of such protocol. Two libraries were found: mz-automation/libiec61850 [42] and ALSETLab/Khorjin-IEC61850-90-5 [3], both available on GitHub. Although, none of them fully address the needs for this research work. The first one is an implementation of IEC 61850 [31], however it doesn't includes a R-GOOSE implementation. On some open topics related to the library, the authors discussed the possibility of including such feature, although, there is no such feature yet.

The second library, Khorjin-IEC61850-90-5, is supposed to fully address the needs of this project, as it implements R-GOOSE and R-SV (Routable-Simple Values), however, the code is not publicly available, as it is still under development. Ultimately, we were able to gain access to an updated version of mz-automation/libiec61850 made by Diogo Saraiva [15] from University of Aveiro, having R-GOOSE already implemented.

R-GOOSE Implementation

Given that, we decided to use the mz-automation/libiec61850 [42] as baseline and perform some changes to it, in order to emulate the generation of R-GOOSE packets. As GOOSE protocol is a Layer 2 protocol, it was expected that the library was generating Ethernet packets. After studying that library and, combined with the study of the R-GOOSE protocol, we add a section on the library to encapsulate the generated GOOSE packet in a UDP packet, adding all of the R-GOOSE headers.

More precisely, we created the UDP payload of the packet, as on the Figure 4.1. On the crafted packet, we inserted all of the fields, starting with the *LI* (0x01), followed by all of the other fixed-values fields. For the R-GOOSE specific fields, like *Session Identifier* and *SPDU Number*, we used a fixed value, in this case the number 1 (one) encoded on the respective number of bytes. For the *Security Information* fields, the approach was the same as the Key Management was outside of the time frame of this research work.

In the next step, we copied the data generated by the library to the remaining, as for example, the *APPID*, *APDU Length* and the *GOOSE PDU*. Finally, we generated the *Signature Field Tag*, with the fixed-value of 0x85, ending the new R-GOOSE packet with the Signature Lenth, in our case, zero.

4.1.2 Devices

As mentioned on the previous section, we decided to focus on Power Distribution System. In this section, we present the analysis made on devices and simulation environments available. Devices are specially important on the Power Distribution Systems environment, as it is important to have in consideration the physical devices existing on real systems, such as IEDs, as main communication source points that use R-GOOSE. In order to do this, there are several possibilities. Can be used real devices, provided by an external entity, or it might be possible to emulate them, by means of hardware and software, such as a Raspberry Pi, or by means of softwarization, using virtual machines or specific software to emulate such devices. For this part, several solutions exist to simulate such devices, although, all of them are proprietary and require a paid license to use. Some of the solution are the follow: 61850 Test Suite Pro, from TriangleMicroworks, Inc. and SmartGridware® IEC 61850 IED Simulator.

In some papers such as [7] and [38], the authors developed IED simulators, one being a low-interaction honey pot. With this said, none of them fully address the necessities of this work, as the first one simulates a real IED, although it is focused on Sample Values. It was developed using LabView and C++. The second one, was developed with the focus on emulating MMS messages, not fully addressing R-GOOSE messages, as it is needed.

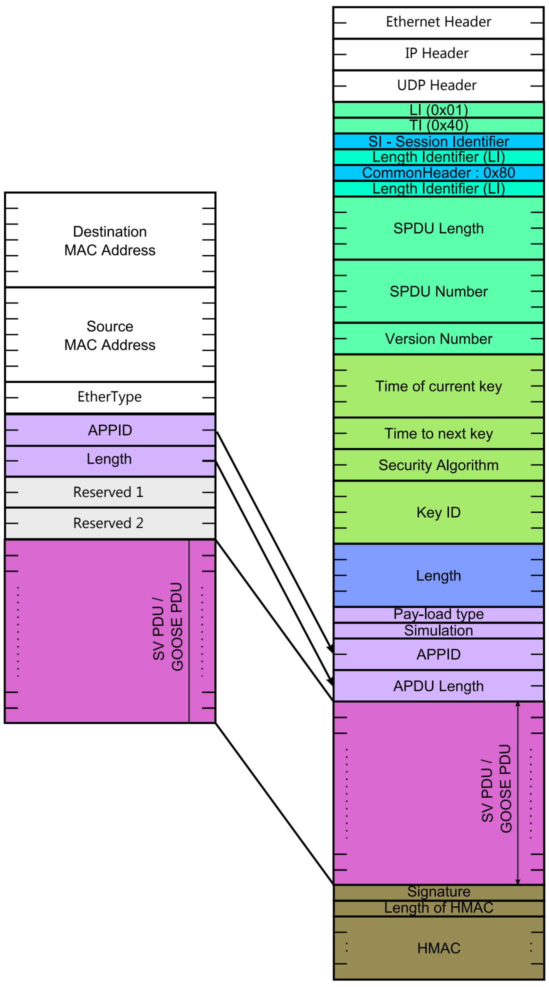


Figure 4.1: Conversion from GOOSE to R-GOOSE

4.2 Proposed Solution

Our goal is to develop a product to provide security mechanisms capable of ensuring a set of common security requirements for critical applications. This set of security requirements was defined in Section 3.2 - Requirements, and through the gap analysis done, we specified that were necessary security mechanisms to ensure Message Integrity, Message Authentication and Message Confidentiality. At the same time, we must guarantee that the new security mechanisms do not compromise the performance of the system.

With that in mind, our proposal is to develop a framework capable of providing such mechanisms to several critical systems. This framework is composed by two major components: a Security Library and a Bridging Device.

The Security Library [16] is a library written in C, that will implement the security mechanisms. In order for this library to be used in several contexts, it is divided in two levels: cryptographic algorithms implementations and integration with protocols. On the first level, a collection of cryptographic algorithms is implemented that can be used to integrate with specific protocols. The integration with protocols uses the cryptographic functions to provide security directly to protocols, dealing with the protocol specific questions. We should note that this library could be used directly on the system devices (like IEDs) or could be used as standalone, as we use on our

Security Gateway.

The Bridging Device [17] will be combined with the Security Library to provide the same security properties to legacy devices or devices without sufficient computational capacity to perform such operations without compromising their performance. This will be called our Security Gateway.

Figure 4.2 illustrates the interactions between both components. There we can see the Security Gateway placed between the device that is intended to be protect (IED) and the network. Also, inside the Security Gateway, we can see represented our Security Library, providing the security mechanisms.

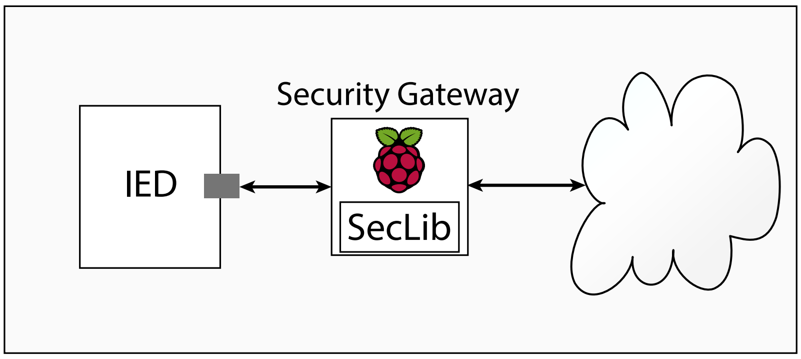


Figure 4.2: Diagram of proposed Security Gateway

In the following sections, both implementations will be described in detail, including the development process and the decisions taken over that process.

4.3 Security Library

The Security Library provides a collection of functions ready-to-use by other applications, to ensure security properties such as integrity, authentication and confidentiality to packets. We can divide the library in two parts: cryptographic algorithms and protocol related functions. The first part contains a set of cryptographic functions, as for example HMAC-SHA256 or AES256-GCM. The second part of the library is responsible to deal with specific aspects of each protocol, as for example, encrypting the packet payload and changing all the other mutable fields of the protocol. In this implementation, we only developed functions to deal with R-GOOSE, being our use case for this research work.

We developed this library to provide three security properties: Message Integrity, Message Authentication and Message Confidentiality. The selection of each algorithm present in the library will be explained later, providing the standard or research work were each algorithm is recommended. The following sections will explain how the library provides each of the properties. Before entering in details, we should note that this library uses OpenSSL 1.1.1 Library, meaning that it is necessary to configure OpenSSL before using our library.

Our library was developed in two versions. The first version was the implementation of cryptographic algorithms and has the following features:

- HMAC Generation functions
- GMAC Generation functions
- AES-GCM encryption and decryption functions

These functions are responsible for performing the cryptographic operations on a given data, producing an HMAC or a GMAC as output, or encrypting and decrypting data. The implementation details of this version are detailed in subsections 4.3.1 and 4.3.2. In the second version of our library we implemented the protocol related functions. These functions use the cryptographic functions developed on the first version of our library to provide the security properties to a given protocol, dealing with the protocol specific aspects, as for example the mutable fields. The implementation details of this version are detailed in subsection 4.3.3 and it has the following features:

- R-GOOSE Authentication
    1. Insert and Validate HMAC
    2. Insert and Validate GMAC
- R-GOOSE Encryption
    1. Encrypt R-GOOSE Payload
    2. Decrypt R-GOOSE Payload

4.3.1 Message Integrity and Message Authentication

To provide Message Integrity and Message Authentication to packets the library provides two type of cryptographic algorithms: HMAC and GMAC. These algorithms receive a chunk of data and a symmetric key, producing a Message Authentication Code. In the case of HMAC, this MAC is generated based on hash generated by other function. In order to provide authentication, the final code is generated using both the message hash and the provided key.

There are several varieties of HMAC depending on the underlying algorithm used to generate the hashs. Choosing this algorithm is very important and scenario dependant, as the hashing algorithm will define both the security of the MAC generated, and the performance efficiency of the HMAC generation.

The GMAC algorithm provides the same properties as HMAC, although it's inner functionalities are different. In this case, the MAC generation is not only based on a hash, but also on a encryption algorithm. GMAC is a particular case of GCM (Galois/Counter Mode) encryption, that is usually used to provided authenticated encryption with associated data. Although, it is possible to use GCM without providing an input to be encrypted but only the data to be authenticated. In that case, it is called GMAC. As it uses a symmetric encryption algorithm, that in our case is AES, it is also necessary to provide an Initialization Vector (IV).

We decided to implement these two algorithms because they are recommended by several standards for communications within Critical Systems, and more precisely, on IEC 62351 [33] for Power Distribution Systems.

For the implementation we used the OpenSSL version 1.1.1 Library. This library already has a base HMAC implementation, as well as a great collection of hashing functions. Furthermore, OpenSSL [45] is a well-known security library, widely used and very tested, representing a safe solution for most of the cryptographic algorithms we need. OpenSSL does not include a full GMAC implementation, however, we used the library to create a GMAC generation function.

As it is detailed by NIST on [53] and by IETF on RFC 4543 [41], to generate a MAC using GMAC, it is necessary to use an encryption algorithm in Galois/Counter Mode (GCM), as for example AES, and provide data to be authenticated without being encrypted. OpenSSL 1.1.1 already implements

AES-GCM making it possible to construct the GMAC algorithm. Using the EVP interface of OpenSSL, we implemented GMAC based on AES.

In the Table 4.1 are shown all of the HMAC variations implemented on our security library, the length in bits of the provided key, and the size in bytes of the produced MAC. We should note that the key and MAC length of each algorithm were recommended on the previously mentioned standards.

Table 4.1: Implemented HMAC-based functions, key size in bits and output size in bytes

HMAC Algorithm	Key size (bits)	MAC size (bytes)	MAC Alg. field value
HMAC-SHA256-80	256	10	0x01
HMAC-SHA256-128	256	16	0x02
HMAC-SHA256-256	256	32	0x03
HMAC-BLAKE2b-80	256	10	0x06
HMAC-BLAKE2s-80	256	10	0x07

In the Table 4.2 are shown the GMAC variations implemented on the library, the length in bits of the provided key, and the size in bytes of the produced MAC. In this case, it must also be provided the IV, although the size is not fixed as it can be specified as an argument on our function.

Table 4.2: Implemented GMAC-based functions, key size in bits and output size in bytes

HMAC Algorithm	Key size (bits)	MAC size (bytes)	MAC Alg. field value
AES128-GMAC-64	128	8	0x08
AES128-GMAC-128	128	16	0x09
AES256-GMAC-64	256	8	0x04
AES256-GMAC-128	256	16	0x05

For the HMAC implementations, we used a function already implemented on OpenSSL [45], the *HMAC()* function. For the GMAC implementations, OpenSSL does not implements a ready-to-use function. As mentioned before, we used the EVP interface to generate the MAC. The EVP interface is a set of functions to create a cryptographic context. Inside this context, we can specify the configurations and operations to be applied on a given input data. In this case, we wanted to use the AES algorithm in GCM mode but without providing any plain text to encrypt, but only data to be authenticated. To do this, we start be creating a new encryption context and defining the encryption algorithm (AES-GCM variant). Then, we use the EVP functions to set the key, the IV and the data to be authenticated. After that, we instruct OpenSSL to perform the encryption and then we extract the MAC from the context. Finally, we truncate the obtained MAC to the necessary length (depending on the algorithm provided by our library).

4.3.2 Message Confidentiality

To provide Message Confidentiality we included a set of AES variants on the library. These variants are all AES in GCM mode, as it is recommended by the IEC 61850 [31] and IEC 6235 [33] standards. Once again, the implementation was made using OpenSSL 1.1.1, using the EVP interface to create both encryption and decryption functions, using AES with several key lengths. Two key size were used, with 128 and 256 bits, and for both of them, an encryption and a decryption function were developed.

The Table 4.3 shows the AES algorithms supported by our security library and their respective key length. As for the Initialization Vector (IV), it can have an arbitrary size.

Table 4.3: Implemented AES-based functions and key size in bits

AES Algorithm	Key size (bits)	Encryption Alg. field value
AES128-GCM-Encrypt	128	0x01
AES128-GCM-Decrypt	128	0x01
AES256-GCM-Encrypt	256	0x02
AES256-GCM-Decrypt	256	0x02

The implementation of the encryption and decryption functions is very similar to the GMAC implementation. We use the EVP interface to create and manage an encryption context. The main difference from the GMAC implementation is that, in this case, we provide the input data to be encrypted/decrypted instead of just being authenticated.

4.3.3 Protocol Related Functions

The second part of the library is a "layer" that will use the cryptographic functions on the packet data. These functions will receive the unsecured protocol packet, and return a secured version. These are the functions that will be applied directly on IEDs and on our bridging device, being used every time a new packet is captured.

In this version of the library, we provide functions to deal with R-GOOSE protocol, following the standards and protocol specification related to security. This set of R-GOOSE related functions include the insertion of an HMAC, the insertion of GMAC, validation of both authentication mechanisms, payload encryption and payload decryption. Furthermore, these functions must address all the mutable fields of the protocol. For example, when adding an HMAC tag to the R-GOOSE protocol, it is necessary to update several length fields and the Security Information fields.

For R-GOOSE communications, we defined that we wanted to ensure Integrity, Authentication and Confidentiality. In order to achieve Integrity and Authentication, we implemented the functions *r_gooseMessage_InsertHMAC* and *r_gooseMessage_InsertGMAC*. These functions are responsible for generating a MAC tag and inserting it on the packet. By the protocol specifications, the entire Session Protocol Data Unit starting from the Session Identifier to the GOOSE Payload should be authenticated. Figure 4.3 illustrates the operations performed by these functions. There we can see that after generating the MAC tag it is inserted at the end of the packet, on the Signature Fields Section. After this, the Security Fields are updated with the information from the key and the algorithms used. This process will ensure message authentication, although we should note that key management is outside of the scope of this research work. Finally, the length fields of the packet are also updated.

To validate such MAC tag, two functions were developed: *r_gooseMessage_ValidateHMAC* and the function *r_gooseMessage_ValidateGMAC* were developed. These functions receive the packet, generate the expected MAC tag using the information from the *Security Fields* and compare it to the MAC tag present in the packet. If they are the same, then the packet is valid, otherwise, the packet is considered invalid.

Confidentiality is provided by the functions *r_gooseMessage_Encrypt* and *r_gooseMessage_Decrypt*. They are responsible for encrypting and decrypting the packets, according to protocol specifications. On an R-GOOSE packet, only the GOOSE PDU should be encrypted. In both encrypting

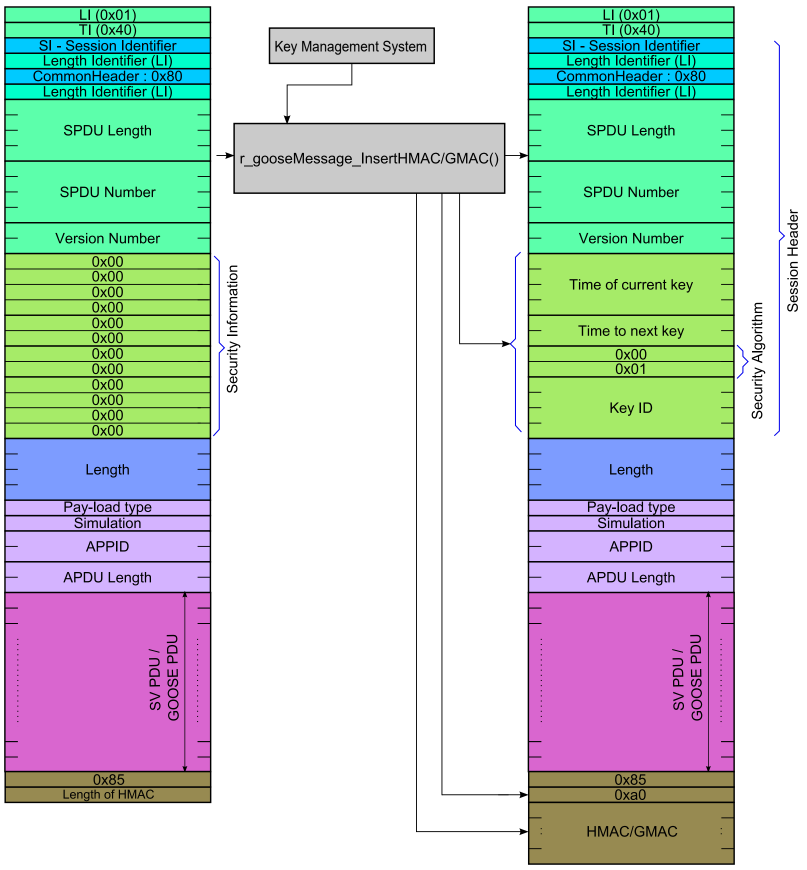


Figure 4.3: Diagram of InsertHMAC/GMAC funtions

and decrypting R-GOOSE packets, the *Security Fields* should also be updated with the used key information and the algorithm used. Figure 4.4 illustrates the operations performed by these functions on R-GOOSE packets.

4.4 Bridging Device

The Bridging Device is the framework component placed between the system component we want to "protect", and the network it is connected to. It will capture the packets sent from the protected device, analyse them and, if necessary, apply the security measures to ensure a given set of security requirements. It will use the Security Library already described to apply the security mechanisms. In terms of hardware, our bridging device is a Raspberry Pi 4B, running Raspbian operating system and equipped with an USB-to-Ethernet adapter, to provide an extra Ethernet port. More precisely, the technical specifications of the Raspberry Pi are presented in the Table 4.4.

This component needs to be particularly efficient, because the total Trip Time of the packets needs to be very low. Using as reference the power distribution system, the Trip Time of the packet must be lower than 3ms (standard IEC 61850 [31]), meaning that capturing, analysing, modifying and sending the packet on the bridge must be performed with extreme efficiency. This component will allow legacy and lower capacity devices to be protected with the necessary security mechanisms, being a component of major important in the context of this work and the 5G Mobilizer project. Also, with this component we can prove that it is possible to use off-the-shelf hardware to improve the security on systems composed by specialized equipments as IEDs,

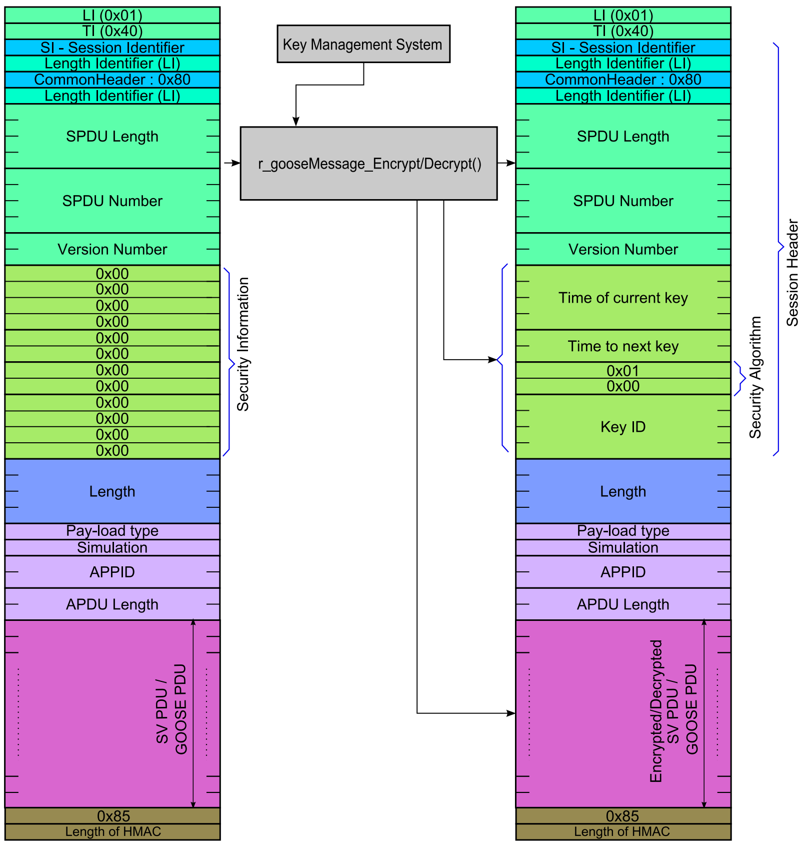


Figure 4.4: Diagram of Encrypt/Decrypt funtions

Table 4.4: Raspberry Pi 4B model Technical Specifications

CPU	Quad core Cortex-A72 (ARM x64) 1.5GHz
RAM	4GB
Ethernet	Gigabit Ethernet
External Ethernet	USB2.0 to Fast Ethernet
Operating System	Raspbian GNU/Linux 10 (Buster)

that usually are not capable of performing such tasks.

4.4.1 Initial Approaches

The development process had several phases, where we first defined a strategy, we implemented a skeleton of that strategy and then we analysed if it was a feasible solution in terms of latency and performance. Our first strategies were based on developing an application that, using sockets, would capture the traffic in promiscuous mode, perform the modifications to the packets and then re-inject them on the network. In brief, the strategy was using the library *Libpcap*. This library provides a set of networking related functions to manipulate network interfaces and capture traffic. This strategy didn't met the performance requirements, as it was taking between 250-750ms to bridge each packet. Our second and third strategies were based on implementing a C application using Raw Sockets where we bridged packets at Layer 3 (second strategy) or at Layer 2 (third strategy). The Layer 3 (or Network Layer) is a layer from the OSI Model that is responsible packet forwarding and routing. The most common protocol used in this layer is the IP. The Layer 2 (or Data link Layer) is a layer from the OSI model that is responsible for the transfer of data between adjacent nodes on a network. In this case, an example of a protocol used in this layer is Ethernet. As an example, an ICMP packet, that is encapsulated inside an IP packet, would be bridged over a Layer 3 bridge, while an ARP packet, that is encapsulated inside an Ethernet frame, wouldn't. Although, in a Layer 2 bridged both of the packets would be bridge because both of them must rely on a Layer 2 protocol as Ethernet. Again, none of this strategies were efficient enough, as they were running with the same latency metrics as *Libpcap*. We understood that to achieve minimum latency on our bridge we won't be able to use normal C applications in user space. With that in mind, we focused on understanding other solutions using features directly from the kernel space.

4.4.2 Linux Bridge and Netfilter

The approach we followed was to create a Linux Bridge handled by the kernel. That bridge would be capturing packets on the incoming interface, then move them from the kernel space to an user space application. This application would modify the packets and re-send them again to the kernel, that in turn would bridge the secured packet to the outgoing interface.

The Linux bridge handled by the kernel was achieved using the *brctl-utils* tool that connects two physical ethernet interfaces creating a new logical interface to the bridge. It is necessary to configure some Linux packages in order to be able to set up such Ethernet Bridge.

After setting up the bridge, it is necessary to move the packets from kernel space to user space. During our research we realized that this could be achieved using IPTables alongside with other Netfilter modules. IPTables is a program that uses several module kernels and the Netfilter kernel framework to configure the networking stack and packet filtering on the kernel, acting as a firewall. It is based on "chains". When a packet arrives to the machine, it is placed in a specific chain, depending on the target of the packet. For example, using the default configurations, if a packet arrives to Machine A with the IP address of that machine, it is placed on INPUT Chain, while if Machine A is routing packets and the packet received is to other machine, then it is placed on FORWARD chain.

A interesting feature of IPTables is that we can set a rule that moves a packet that arrives in a given chain to a NFQUEUE for a custom packet filtering by an user space application. This is a functionality of Netfilter that allows a user to develop custom filtering functions in user space, giving the power to that application to set the verdict on a given packet. Even if it is not a default behavior for IPTables and Netfilter, it is possible to modify or even craft a new packet at user space and send it to kernel space, using the same interface.

However, IPTables rules are not applied by default on Linux bridges, because Linux bridge works at Layer 2 and IPTables deal with Layer 3 packets. Nevertheless, we can change this behavior using a Kernel Module. The *br_netfilter* Kernel Module allows the IPTables to filter bridged packets from a Linux Bridge, where we can just use the FORWARD Chain to collect such packets.

After setting the Linux Bridge and loading the *br_netfilter* Kernel Module it is necessary to create an IPTables rule to move the packets arriving at our bridge to an user space application. Those packets are placed on the FORWARD Chain of IPTables. Given that, we will add a new rule to the FORWARD Chain, using the argument "-j NFQUEUE".

Finally, the user space application will be listening for packets moved by IPTables to the NFQUEUE. Netfilter provides a library to manage such queues, being handled the same way as sockets. The Algorithm 1 illustrates how the user space applications was implemented. To collect packets

Algorithm 1 User Space Application

```
1: procedure MAIN(configurations)
2:     handler ← obtain socket file descriptor (nfq_open)
3:     bind handler to NFQUEUE (nfq_create_queue)
4:     while true do
5:         buffer ← receive packet (nfq_handle_packet)
6:         Call callback(buffer,handler)
7:     close
8: procedure CALLBACK(packet,handler)
9:     if packet.protocol = configurations.transport_protocol then
10:        if packet.destination_port = configurations.destination_port then
11:            if packet.protocol_specific_fields = configurations.protocol_specific_fields then
12:                if configurations.encryption = true then
13:                    buffer ← Encryption function
14:                    Call setVerdict(buffer, verdict)
15:                if configurations.authentication = true then
16:                    buffer ← Authentication function
17:                    Call setVerdict(buffer, verdict)
18:            else
19:                Call setVerdict(buffer, REJECT)
20:        else
21:            Call setVerdict(buffer, ACCEPT)
22:    else
23:        Call setVerdict(buffer, ACCEPT)
24:    close
```

placed in NFQUEUEs, first it is necessary to obtain a file descriptor to the queue, using the library provided by Netfilter. Then, we implemented a loop cycle using the *recv* function on the file descriptor previously obtained. When a packet is received, we call a callback function to process each packet. This is illustrated in pseudo-code on the procedure Main of the Algorithm 1.

Inside the callback function each packet is processed. This function can be customized and designed to fit any other protocol. In our case, we created a set of configurations to process each R-GOOSE packet namely, interface1 and interface2, the port where R-GOOSE is running, the authentication algorithm and the encryption algorithm. In the Algorithm 1, this configurations are represented by the variable configurations. The interfaces are used to distinguished the communication flow. For example, if a packet arrives from the IED, we may want to add an HMAC, although if a packet arrives from the network we must verify the HMAC tag it contains. The port is used to identify what packets are indeed R-GOOSE packets. Finally, the authentication and encryption algorithms specify which algorithms will be used to encrypt or generate the HMAC, although, they can be set to "None", skipping authentication or encryption.

The first thing we compare when addressing the R-GOOSE protocol is the transport protocol of the packet. In this case, it must be UDP as R-GOOSE is sent over UDP (Algorithm 1, line 9). After that, we analyse the destination port of the packet and we compare to the one that we configured R-GOOSE to use (Algorithm 1, line 10). The next verification compares some fixed-value fields of the packet, for example the initialization tags of the Session Protocol Data Unit and the Session Header Length, in order to validate that we are indeed dealing with an R-GOOSE packet (Algorithm 1, line 11). If all of the verifications are succeed, we use the Security Library to apply the required security mechanisms that are defined on the configurations (Algorithm 1, lines 12 - 17). If only the specific protocol fields verification fails, then the packet is rejected as it has malformed fields (Algorithm 1, lines 18 and 19). Otherwise, the packet is accepted as it is not an R-GOOSE packet (Algorithm 1, lines 20 - 23). To reject, accept or re-inject a packet we use the function *setVerdict* from the Netfilter NFQUEUE library (Algorithm 1, lines 14, 17, 19, 21 and 23). This function sends the verdict to IPTables and re-inserts the provided packet (*buffer* on the Algorithm 1) on the NFQUEUE in use.

When a valid R-GOOSE packet is identified and if the security mechanism we are going to apply will modify the packet size, then we make all of the changes in a copied version of the packet and we send this new crafted packet to the kernel. If not, then all of the changes are made directly on the original packet. To send the packet back to the kernel space, we just need to set a verdict on that packet. This verdict will signalize IPTables that we already decided if we want or not to accept the packet. In our case, accepting the packet means that it will proceed through the bridge.

The Figure 4.5 illustrates our bridging solution, showing in detail the interactions between IPTables, NFQUEUE and our user space program. The blue arrows represent the path that an arriving packet will do, while the green arrow represents the verdict issued by our program to IPTables. In this example, the packet was accepted, although we can also reject the packet, leading IPTables to drop it.

4.5 Chapter Wrap-up

This chapter covers the initial experiments, the solution proposed and it's implementation. Firstly, we analysed the available open-source implementations of protocols and devices that could be used on the contexts of the 5G Mobilizer project applications. This was a challenging task as there were no available implementations for protocols in Railway Signalling systems and PPDR

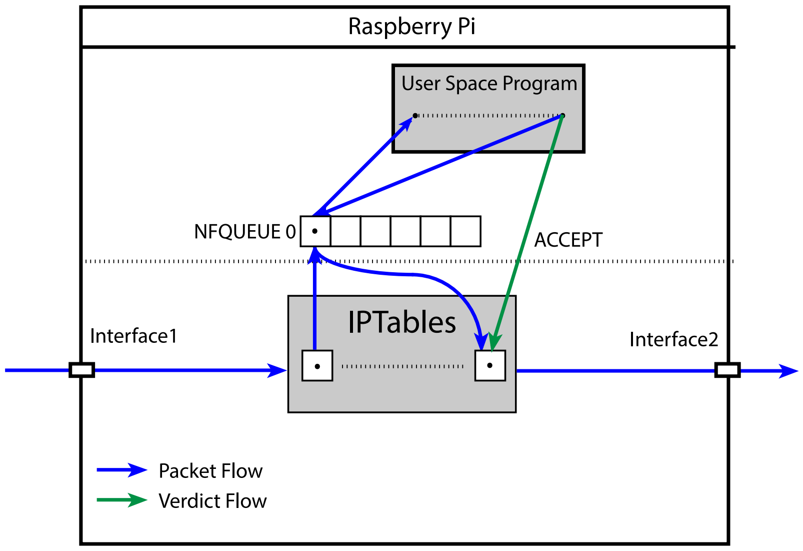


Figure 4.5: Bridge Diagram - Interaction between IPTables, Netfilter NFQUEUEs and user space program in a one way communication

platforms. For the Power Distribution system we found a GOOSE implementation and performed several modifications in order to be able to send R-GOOSE packets.

Secondly, we presented our solution. Our solution has two components, a security library and bridging device that combined form our Security Gateway. This library is written in C and contains a set of cryptographic functions and several protocol integration functions. As we only had access to the R-GOOSE implementation, we only developed functions to support this protocol. In terms of the security mechanisms for R-GOOSE, we developed functions to ensure message authentication, message integrity and message confidentiality. The bridging device is a Raspberry Pi 4B specially configured to bridge and modify packets. To achieve this, we created a Linux Bridge using *brctl-utils* alongside with a Netfilter Kernel Module to move packets to a user space application, using IPTables. In that user space application, we filter R-GOOSE packets and we apply the security mechanisms developed on our security library to them. The advantages of this solution is that with off-the-shelf hardware we can provide security mechanisms to specialized devices that are not capable of performing such task, as legacy or low capability devices. In the next chapter we will present and analysed the obtained results to evaluate the performance of our solution.

Chapter 5

Results and Analysis

In this chapter, we present and analyse the obtained results from the practical evaluation. In short, we focused on studying if the developed mechanisms were able to provide the security requirements (Section 3.2) without breaking the performance requirements. Following the implementation work (Chapter 4), we used the Power Distribution Systems as our practical use case. In terms of security requirements, are goal is to provide integrity, authentication and confidentiality to R-GOOSE messages. In terms of performance requirements, we want our Transfer Time (Figure 3.4) lower than 3ms, as specified in IEC 61850 [31]. On the next section, we will present our evaluation strategy, detailing the experimental scenario and the tests performed. In Sections 5.2 and 5.3, we present and analyse the results of each test.

5.1 Evaluation Strategy

Our evaluation strategy was focused in validating the security provided by our mechanisms and the impact that they have in the system. We focused our experiments in two components: Security Library itself and the Raspberry Pi Bridge. The purpose of evaluating the Security Library by itself was to analyse its impact when incorporated in an IED or other dedicated device. To perform such evaluation we ran our library only on the Raspberry Pi, as it is has lower computational capacities than normal PCs, as IEDs.

The goal of evaluating the Raspberry Pi Bridging Device was to analyse the impact of introducing a new component into the system. In order to achieve that, we set up an experimental environment to simulate the communications between two nodes of a Power Distribution System. Our experimental environment is composed by two PCs representing endpoint nodes of the network, and our Raspberry Pi. All of the components are connect by Ethernet cables, as shown in Figure 5.1.

Some software components were developed to help in traffic simulation and metrics collection. The libIEC61850 [42] was modified as explained in Section 4.1.1, allowing the generation of R-GOOSE traffic. Finally, an application was developed as an UDP Server, receiving R-GOOSE packets and re-sending them to the source IP Address with the objective of measuring the communications Round Trip Time (RTT).

Table 5.1 presents the specifications for the components illustrated in Figure 5.1 (Raspberry Pi Bridging Device specifications can be found in Section 4.3).

We organized our evaluation in two parts: Functional Evaluation and Performance Evaluation. In the functional evaluation, we performed several tests to evaluate the correctness and effective-

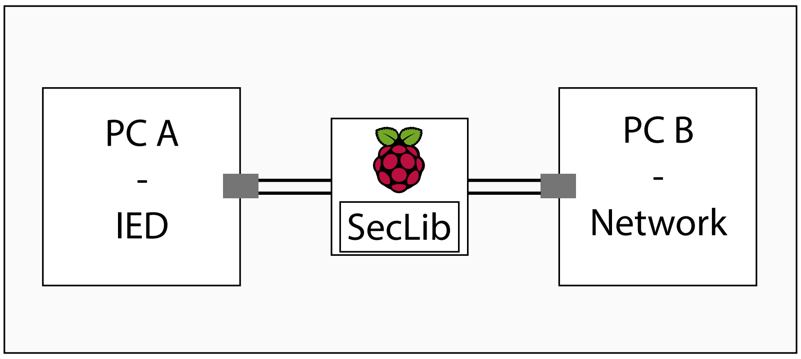


Figure 5.1: Raspberry Pi Bridging Device Experimental Environment

Table 5.1: Raspberry Pi 4B model Technical Specifications

Component	**PC A**	**PC B**
CPU	Intel Core Duo E6400 @ 2.13GHz	Quad core i7-6500U CPU @ 2.50GHz
RAM	4GB	16GB
Ethernet	Gigabit Ethernet	Gigabit Ethernet
Operating System	Ubuntu 16.04 (Xenial)	Windows 10

ness of our security mechanisms. For example, we analysed if the output of the cryptographic functions was correct and if it was correctly placed inside the R-GOOSE messages. In the performance evaluation, we focused on measuring the impact that our mechanisms had on the system, specially in terms of latency. We started by evaluating the Security Library. By doing this, we managed to select the best and the worst algorithms, in terms of performance, to be applied on the bridging device, allowing us to define the boundaries of the latency our solution will introduce.

On the Functional Evaluation phase we performed the following experiments:

1. **Cryptographic Functions** - Provide a given input based on standardized test vectors from RFCs, to the cryptographic function and analyse the output, comparing with the expected. All of the developed functions were tested.

2. **Protocol Related Functions** - Provide an unsecured R-GOOSE packet and analyse the output. On the output, we analysed if the structure and the mutable fields were properly updated, having in consideration the protocol specifications. All of the developed functions were tested using only one cryptographic function.

For the Performance Evaluation phase we performed the following experiments:

1. **Cryptographic Functions** - Set a timer before the function execution, provide an input to the cryptographic function and set a timer at the end of its execution, measuring the difference between them. All of the developed functions were tested.

2. **Protocol Related Functions** - Set a timer before the function execution, provide an unsecured R-GOOSE packet to the function and set a timer at the end of its execution, measuring the difference between them. All of the developed functions were tested, using all of the cryptographic functions developed.

3. **Raspberry Pi Bridging Device** - To properly evaluate the bridging device performance, we evaluated measured the following metrics:

    (a) Communications Bandwidth

    (b) Traffic Latency

4. **Security Gateway** - Generate R-GOOSE traffic on the experimental scenario and analyse the communications latency using the Security Gateway

    (a) R-GOOSE Traffic Latency - Only tested the best and worst performing cryptographic functions, using the values measured from the protocol related functions in point 2 above.

On the next sections, we will present and analyse in detail the experimentation results from Functional Evaluation (Section 5.2) and Performance Evaluation (Section 5.3).

5.2 Functional Evaluation

In this section, we will describe the method we applied to evaluate the functional requirements and correctness of the mechanisms we developed. Firstly, we will present the Security Library evaluation, starting with the cryptographic functions and followed by the protocol related functions. Then, we selected one cryptographic algorithm per protocol related function (for HMAC insertion and validation, GMAC insertion and validation, payload encryption and decryption).

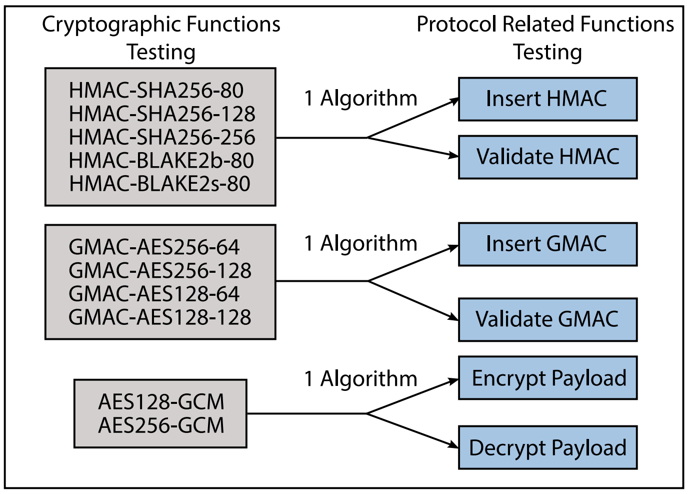


Figure 5.2: Functional Experimentation - Relationship between cryptographic functions and protocol related functions in testing

This algorithm will be used to evaluate each protocol related function (Figure 5.2), where only 1 algorithm of the cryptographic functions is used at a time in the protocol related functions. All of these tests were performed only on the Raspberry Pi.

If the functional validation of our Security Library is positive, then the functional validation of the Security Gateway, in terms of ensuring security requirements to R-GOOSE packets, is also guaranteed, since the same version is used.

5.2.1 Cryptographic Functions

The Cryptographic Functions evaluation was performed by supplying a given input to the unitary functions and analyse the output produced. Using the function *hmac_SHA256_80* as an example, we provided a key and an input to be authenticated. Then, we analysed if the generated HMAC was equal to the expected one, as shown in the Figure 5.3

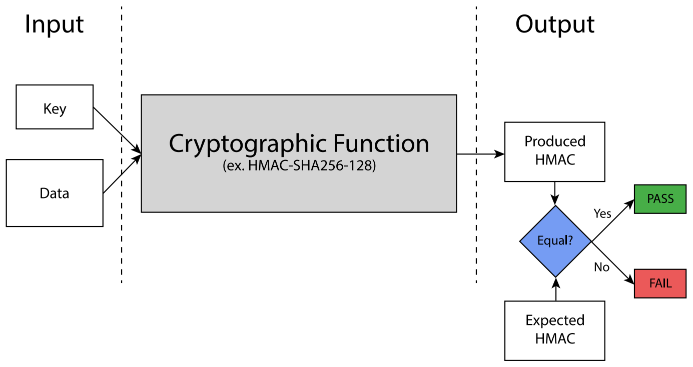


Figure 5.3: Example of Functional Testing procedure for Cryptographic Functions

In our library, we have three types of cryptographic functions: HMAC generation, GMAC generation and encryption/decryption using AES-GCM. In order to properly evaluate these functions, we used standardized test vectors. In the case of HMAC generation, we used a set of test vectors from the RFC 4231 [43] and RFC 7693 [6]. The first RFC contains the test vectors for HMAC-SHA-based, while the second contains the test vectors for HMAC-BLAKE-based.

In the case of GMAC generation and encryption/decryption using AES-GCM, we used a set of test vectors produced by NIST issued on the Special Publication 800-38D [53]. Attached to the document, several files contain multiple test vectors for both plain text encryption/decryption with AES-GCM and data authentication with AES-GCM, in other words, GMAC. Those files contain around 8250 test vectors, of which only 1260 are related with GMAC, using different key and data sizes.

In the Tables 5.2, 5.3 and 5.4, we present the results from the functional evaluation of the cryptographic functions:

Table 5.2: HMAC Functions evaluation results

Function	**Test Vectors**	**Result**
HMAC-SHA256-80	RFC 4321	**PASS**
HMAC-SHA256-128	RFC 4321	**PASS**
HMAC-SHA256-256	RFC 4321	**PASS**
HMAC-BLAKE2b-80	RFC 7693	**PASS**
HMAC-BLAKE2s-80	RFC 7693	**PASS**

Analysing the tables we can see that the output produced by our functions corresponded to the expected in all the test cases. As for the inner security of each algorithm, it is guaranteed by the usage of OpenSSL, which provides a good level of confidence.

Table 5.3: GMAC Functions evaluation results

Function	**Test Vectors**	**Result**
GMAC-AES256-64	NIST SP 800-38D	**PASS**
GMAC-AES256-128	NIST SP 800-38D	**PASS**
GMAC-AES128-64	NIST SP 800-38D	**PASS**
GMAC-AES128-128	NIST SP 800-38D	**PASS**

Table 5.4: AES Encrypt/Decrypt Functions evaluation results

Function	**Test Vectors**	**Result**
AES128-GCM-Encrypt	NIST SP 800-38D	**PASS**
AES128-GCM-Decrypt	NIST SP 800-38D	**PASS**
AES256-GCM-Encrypt	NIST SP 800-38D	**PASS**
AES256-GCM-Decrypt	NIST SP 800-38D	**PASS**

5.2.2 Protocol Related Functions

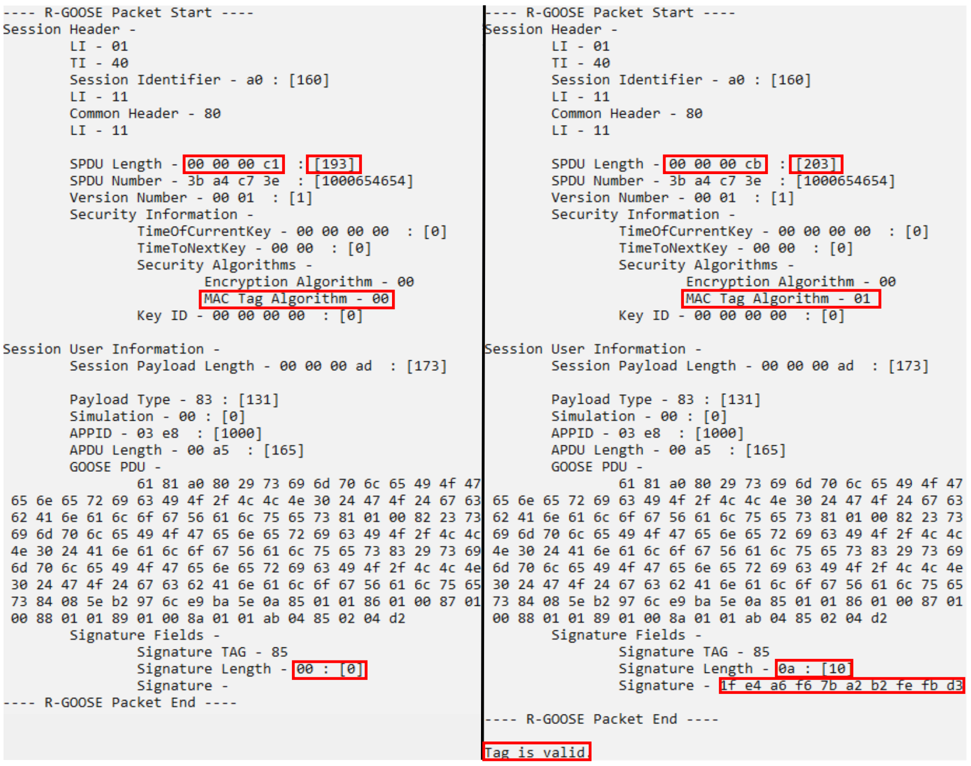
The evaluation of Protocol Related Functions was similar to the Cryptographic Functions. In this case, the input we supplied to the functions was an R-GOOSE packet generated with the adaptations we made on *libIEC61850* [42]. After the execution of the function, we compared the resulting packet with the expected packet. As an example, when inserting an HMAC on an R-GOOSE using the function *r_gooseMessage_InsertHMAC*, we analyse if the HMAC was inserted on the right place and if all of the mutable fields were correctly changed and calculated.

In this phase of the functional testing, we were not concerned with the MAC tag itself, but we were focused on understanding if all of the changes were correctly performed and accordingly to the protocol specifications [31]. Given that, we selected only one cryptographic algorithm for each protocol related function to be used during these tests. To analyse the changes made by each function, we developed a function that dissects each packet and displays the information in a user friendly way.

We used the same methodology to evaluate HMAC and GMAC functions, and both insertion and validation were done in the same tests. For HMAC/GMAC insertion, our functions must insert the HMAC/GMAC, change signature size, update the Security Fields values and update the SPDU Length. For the validation, the function doesn't makes any change on the packet, although, it evaluates the HMAC/GMAC on the packet with the HMAC/GMAC calculated by the function, from the packet content and the key associated to the packet.

In the Figure (5.4) we demonstrate the application of *InsertHMAC* and *ValidateHMAC* on a R-GOOSE packet. There we can verify that our function is properly modifying the mutable fields of the protocol according to the specifications. We must note that in this case only the MAC Algorithm of the Security Fields was changed because we didn't provide the other fields, as there is no Key Management System in place. We can also see that the function *ValidateHMAC* is working properly. In this case, we add the HMAC to the R-GOOSE packet and immediately after we called the validation function, that evaluated as a valid packet.

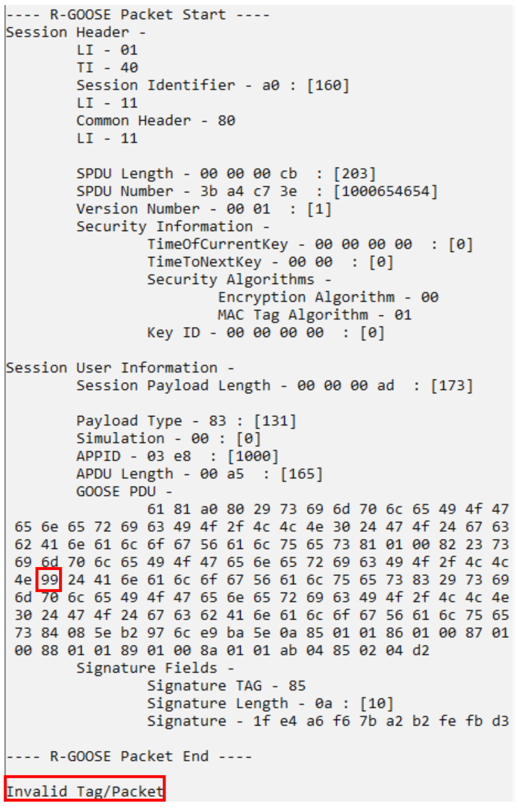
In the Figure (5.5) is the output of ValidateHMAC function when we deliberately modified the packet after the HMAC insertion, simulating an attack. The first red square, the "99" value, is the modified byte after the HMAC insertion. The red rectangle at the end shows that the validation function invalidated that packet, as expected.



```
---- R-GOOSE Packet Start ----
Session Header -
        LI - 01
        TI - 40
        Session Identifier - a0 : [160]
        LI - 11
        Common Header - 80
        LI - 11

        SPDU Length - 00 00 00 c1  : [193]
        SPDU Number - 3b a4 c7 3e  : [1000654654]
        Version Number - 00 01  : [1]
        Security Information -
                TimeOfCurrentKey - 00 00 00 00  : [0]
                TimeToNextKey - 00 00  : [0]
                Security Algorithms -
                        Encryption Algorithm - 00
                        MAC Tag Algorithm - 00
                Key ID - 00 00 00 00  : [0]

Session User Information -
        Session Payload Length - 00 00 00 ad  : [173]

        Payload Type - 83 : [131]
        Simulation - 00 : [0]
        APPID - 03 e8  : [1000]
        APDU Length - 00 a5  : [165]
        GOOSE PDU -
                61 81 a0 80 29 73 69 6d 70 6c 65 49 4f 47
 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e 30 24 47 4f 24 67 63
 62 41 6e 61 6c 6f 67 56 61 6c 75 65 73 81 01 00 82 23 73
 69 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c
 4e 30 24 41 6e 61 6c 6f 67 56 61 6c 75 65 73 83 29 73 69
 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e
 30 24 47 4f 24 67 63 62 41 6e 61 6c 6f 67 56 61 6c 75 65
 73 84 08 5e b2 97 6c e9 ba 5e 0a 85 01 01 86 01 00 87 01
 00 88 01 01 89 01 00 8a 01 01 ab 04 85 02 04 d2
        Signature Fields -
                Signature TAG - 85
                Signature Length - 00 : [0]
                Signature -
---- R-GOOSE Packet End ----
```

```
---- R-GOOSE Packet Start ----
Session Header -
        LI - 01
        TI - 40
        Session Identifier - a0 : [160]
        LI - 11
        Common Header - 80
        LI - 11

        SPDU Length - 00 00 00 cb  : [203]
        SPDU Number - 3b a4 c7 3e  : [1000654654]
        Version Number - 00 01  : [1]
        Security Information -
                TimeOfCurrentKey - 00 00 00 00  : [0]
                TimeToNextKey - 00 00  : [0]
                Security Algorithms -
                        Encryption Algorithm - 00
                        MAC Tag Algorithm - 01
                Key ID - 00 00 00 00  : [0]

Session User Information -
        Session Payload Length - 00 00 00 ad  : [173]

        Payload Type - 83 : [131]
        Simulation - 00 : [0]
        APPID - 03 e8  : [1000]
        APDU Length - 00 a5  : [165]
        GOOSE PDU -
                61 81 a0 80 29 73 69 6d 70 6c 65 49 4f 47
 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e 30 24 47 4f 24 67 63
 62 41 6e 61 6c 6f 67 56 61 6c 75 65 73 81 01 00 82 23 73
 69 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c
 4e 30 24 41 6e 61 6c 6f 67 56 61 6c 75 65 73 83 29 73 69
 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e
 30 24 47 4f 24 67 63 62 41 6e 61 6c 6f 67 56 61 6c 75 65
 73 84 08 5e b2 97 6c e9 ba 5e 0a 85 01 01 86 01 00 87 01
 00 88 01 01 89 01 00 8a 01 01 ab 04 85 02 04 d2
        Signature Fields -
                Signature TAG - 85
                Signature Length - 0a : [10]
                Signature - 1f e4 a6 f6 7b a2 b2 fe fb d3

---- R-GOOSE Packet End ----

Tag is valid
```

Figure 5.4: Functional testing of HMAC insertion and validation

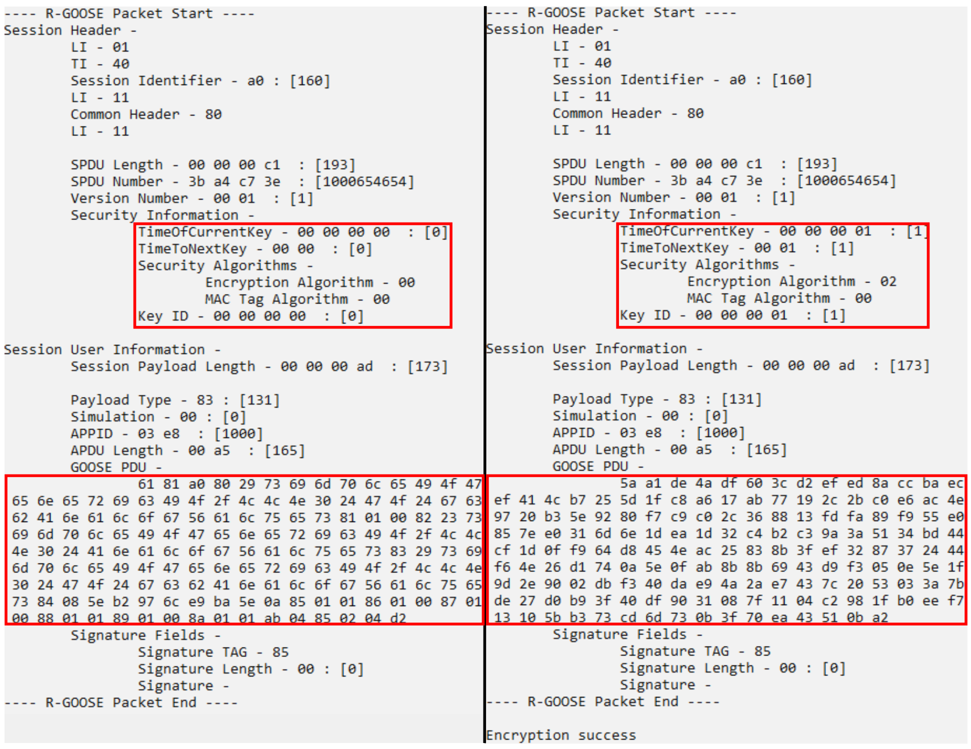


```
---- R-GOOSE Packet Start ----
Session Header -
        LI - 01
        TI - 40
        Session Identifier - a0 : [160]
        LI - 11
        Common Header - 80
        LI - 11

        SPDU Length - 00 00 00 cb  : [203]
        SPDU Number - 3b a4 c7 3e  : [1000654654]
        Version Number - 00 01  : [1]
        Security Information -
                TimeOfCurrentKey - 00 00 00 00  : [0]
                TimeToNextKey - 00 00  : [0]
                Security Algorithms -
                        Encryption Algorithm - 00
                        MAC Tag Algorithm - 01
                Key ID - 00 00 00 00  : [0]

Session User Information -
        Session Payload Length - 00 00 00 ad  : [173]

        Payload Type - 83 : [131]
        Simulation - 00 : [0]
        APPID - 03 e8  : [1000]
        APDU Length - 00 a5  : [165]
        GOOSE PDU -
                61 81 a0 80 29 73 69 6d 70 6c 65 49 4f 47
 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e 30 24 47 4f 24 67 63
 62 41 6e 61 6c 6f 67 56 61 6c 75 65 73 81 01 00 82 23 73
 69 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c
 4e 99 24 41 6e 61 6c 6f 67 56 61 6c 75 65 73 83 29 73 69
 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e
 30 24 47 4f 24 67 63 62 41 6e 61 6c 6f 67 56 61 6c 75 65
 73 84 08 5e b2 97 6c e9 ba 5e 0a 85 01 01 86 01 00 87 01
 00 88 01 01 89 01 00 8a 01 01 ab 04 85 02 04 d2
        Signature Fields -
                Signature TAG - 85
                Signature Length - 0a : [10]
                Signature - 1f e4 a6 f6 7b a2 b2 fe fb d3

---- R-GOOSE Packet End ----

Invalid Tag/Packet
```

Figure 5.5: ValidateHMAC function invalidating a packet

For the functional validation of encryption and decryption, we gave an R-GOOSE packet as input and we analysed the resulting packet. The encryption/decryption function should encrypt or decrypt the GOOSE Payload of the packet and change the Security Information fields with the encryption algorithm and the provided key details.

In the Figure (5.6) is illustrated the encryption of an R-GOOSE packet. In this example, we provided forged information related to the key used. The first red box highlights the changes made on the Security Information fields, while second box highlights the encrypted/decrypted payload.



```
---- R-GOOSE Packet Start ----
Session Header -
        LI - 01
        TI - 40
        Session Identifier - a0 : [160]
        LI - 11
        Common Header - 80
        LI - 11

        SPDU Length - 00 00 00 c1  : [193]
        SPDU Number - 3b a4 c7 3e  : [1000654654]
        Version Number - 00 01  : [1]
        Security Information -
                TimeOfCurrentKey - 00 00 00 00  : [0]
                TimeToNextKey - 00 00  : [0]
                Security Algorithms -
                        Encryption Algorithm - 00
                        MAC Tag Algorithm - 00
                Key ID - 00 00 00 00  : [0]

Session User Information -
        Session Payload Length - 00 00 00 ad  : [173]

        Payload Type - 83 : [131]
        Simulation - 00 : [0]
        APPID - 03 e8  : [1000]
        APDU Length - 00 a5  : [165]
        GOOSE PDU -
                61 81 a0 80 29 73 69 6d 70 6c 65 49 4f 47
 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e 30 24 47 4f 24 67 63
 62 41 6e 61 6c 6f 67 56 61 6c 75 65 73 81 01 00 82 23 73
 69 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c
 4e 30 24 41 6e 61 6c 6f 67 56 61 6c 75 65 73 83 29 73 69
 6d 70 6c 65 49 4f 47 65 6e 65 72 69 63 49 4f 2f 4c 4c 4e
 30 24 47 4f 24 67 63 62 41 6e 61 6c 6f 67 56 61 6c 75 65
 73 84 08 5e b2 97 6c e9 ba 5e 0a 85 01 01 86 01 00 87 01
 00 88 01 01 89 01 00 8a 01 01 ab 04 85 02 04 d2
        Signature Fields -
                Signature TAG - 85
                Signature Length - 00 : [0]
                Signature -
---- R-GOOSE Packet End ----
```

```
---- R-GOOSE Packet Start ----
Session Header -
        LI - 01
        TI - 40
        Session Identifier - a0 : [160]
        LI - 11
        Common Header - 80
        LI - 11

        SPDU Length - 00 00 00 c1  : [193]
        SPDU Number - 3b a4 c7 3e  : [1000654654]
        Version Number - 00 01  : [1]
        Security Information -
                TimeOfCurrentKey - 00 00 00 01  : [1]
                TimeToNextKey - 00 01  : [1]
                Security Algorithms -
                        Encryption Algorithm - 02
                        MAC Tag Algorithm - 00
                Key ID - 00 00 00 01  : [1]

Session User Information -
        Session Payload Length - 00 00 00 ad  : [173]

        Payload Type - 83 : [131]
        Simulation - 00 : [0]
        APPID - 03 e8  : [1000]
        APDU Length - 00 a5  : [165]
        GOOSE PDU -
                5a a1 de 4a df 60 3c d2 ef ed 8a cc ba ec
 ef 41 4c b7 25 5d 1f c8 a6 17 ab 77 19 2c 2b c0 e6 ac 4e
 97 20 b3 5e 92 80 f7 c9 c0 2c 36 88 13 fd fa 89 f9 55 e0
 85 7e e0 31 6d 6e 1d ea 1d 32 c4 b2 c3 9a 3a 51 34 bd 44
 cf 1d 0f f9 64 d8 45 4e ac 25 83 8b 3f ef 32 87 37 24 44
 f6 4e 26 d1 74 0a 5e 0f ab 8b 8b 69 43 d9 f3 05 0e 5e 1f
 9d 2e 90 02 db f3 40 da e9 4a 2a e7 43 7c 20 53 03 3a 7b
 de 27 d0 b9 3f 40 df 90 31 08 7f 11 04 c2 98 1f b0 ee f7
 13 10 5b b3 73 cd 6d 73 0b 3f 70 ea 43 51 0b a2
        Signature Fields -
                Signature TAG - 85
                Signature Length - 00 : [0]
                Signature -
---- R-GOOSE Packet End ----

Encryption success
```

Figure 5.6: Encryption of an R-GOOSE packet

The Table 5.5 summarizes the tests and respective result on each Protocol Related Function developed.

Table 5.5: Protocol Related Functions evaluation results

Function	**Result**
R-GOOSE_InsertHMAC	**PASS**
R-GOOSE_ValidateHMAC	**PASS**
R-GOOSE_InsertGMAC	**PASS**
R-GOOSE_ValidateGMAC	**PASS**
R-GOOSE_Encrypt	**PASS**
R-GOOSE_Decrypt	**PASS**

After analysing all of the functions in terms of security and functionality, we can state that all of them achieved their objective, passing all of the tests performed.

5.3 Performance Evaluation

In this section, we will describe the method we applied to evaluate the performance of each component we developed. Firstly, we will present the Security Library evaluation, followed by the Raspberry Pi Bridging Device evaluation and then the combination of both components, the Security Gateway.

We will start by present the performance evaluation of the cryptographic functions (Section 5.3.1), where all were tested. Then, we analyse the performance of the Protocol Related Functions. Here we analyse all of the combinations of protocol related function with cryptographic function (Section 5.3.2). In other words, we fully tested the performance of the Security Library. These test were only run on the Raspberry Pi. We defined that the execution of the protocol related functions should take less than 0.3ms, corresponding to 10% of the maximum Transfer Time allowed by the IEC 61850 standard [31]. This value allows us to have a sufficient time window left for the bridging process and trip time of data.

The second component evaluated was the Raspberry Pi as a bridging device only. With this stack of tests, we tried to understand if this component, acting as a simple bridge, would comply with the performance requirements of critical systems. In order to achieve this, we evaluated two metrics: bandwidth of the communications passing through the bridge, and latency it would add to the communications.

Our final goal was to analyse if both components combined, acting as a Security Gateway, were a feasible solution in terms of performance. This final tests must comply with the performance requirements collected in Section 3.2.2. The testing files used, the produced outputs and the raw tables used to calculate the metrics presented can be found in the Security Library Github repository [16], inside the *test* directory. Also, the complete testing tables for the Security Library can be found on Appendix A.

5.3.1 Cryptographic Functions

Similar to the way we did the functional evaluation, here we perform this evaluation by supplying an input to each unitary function. However, we were not focused on the output of the function, instead we measured the time, in milliseconds, that the function took to produce the output. This measurement was made using monotonic clock of the computer, setting a timer before the execution, and another timer at the end. Then, the execution time is obtained by subtracting the first value from the second value (Figure 5.7).

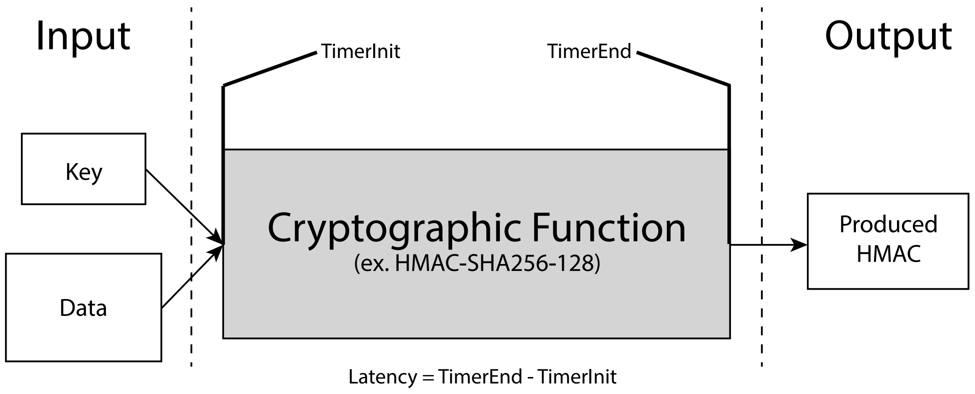


Figure 5.7: Example of Performance Testing procedure for Cryptographic function

To present a valid value for the latency, we repeated each experiment 500 000 times. From that dataset, we calculated the average latency, the standard deviation, maximum and minimum value, and the 95% confidence interval. Also, we considered three input sizes to cover several operation scenarios. Those data sizes were obtained by analysing several R-GOOSE messages. The first input size was 196 bytes, corresponding to a small R-GOOSE message. The second data size was 256 bytes, corresponding to an intermediate R-GOOSE message. Lastly, the third data size was 572 bytes, corresponding to a intermediate/large R-GOOSE message. We chosen these values after analysing several R-GOOSE messages generated with libIEC61850 [42]. The first data size corresponds to a message containing 1 GOOSE object, while the second data size corresponds to a message containing 12 GOOSE objects, and the last data size corresponds to a message containing 72 GOOSE objects. Using three different data sizes will also allow us to understand how the latency varies with the size of the input, that later could be used to decide which algorithm should be used in each scenario.

Table 5.6: HMAC Functions performance analysis, using medium size data input

HMAC Algorithm	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
HMAC-SHA256-80	0.007	0.001	0.249	0.007	± 0.002
HMAC-SHA256-128	0.007	0.001	0.249	0.007	± 0.002
HMAC-SHA256-256	0.007	0.001	0.301	0.007	± 0.002
HMAC-BLAKE2b-80	0.012	0.003	0.783	0.012	± 0.004
HMAC-BLAKE2s-80	0.008	0.001	0.254	0.008	± 0.002

Analysing the Table 5.6, we can see that all of the HMAC algorithms met the established requirement for time execution. We can see that all of the SHA256 variations performed with very similar times. It was expected, as the only difference between them is the size of the output and the number of bytes copied. On the other hand, it is clear that BLAKE2b didn't perform as well as SHA256. As BLAKE2 variants were designed to achieve high performance, an explanation to this behavior could be that these algorithms are not so optimized as SHA256 in OpenSSL Library [45]. We can point out that BLAKE2b-80 clearly was the worst performing algorithm. Analysing the confidence intervals, they are all similar to all algorithms with some exceptions. In the case of small input size, HMAC-SHA256-80 had a slightly higher value. In the case of medium input size, HMAC-BLAKE2b-80 also had a slightly higher confidence interval, and for the large input size we should note that the confidence interval of HMAC-BLAKE2s is higher than the rest. Also, in all of this cases, the maximum is higher than the others. This behavior can be explained by the presence of outliers. Finally, we must note that the maximum value is much higher than the average, although it is possibly due to other processes running on the Raspberry Pi in simultaneous. Yet, these values were still lower than the targeted 0.3ms.

On the Figure 5.8, we illustrate the data collected from testing with different input data sizes. From the graphic, we can also directly compare the performance of each algorithm for a specific data size. First, we should note that green, blue and red plots have very similar values, being overlapping in the graphic. Secondly, we can see that for all data inputs, the best performing algorithms are the SHA256 variants. As for BLAKE2s-80, it has similar performance to SHA256 variants, although it is slightly worst performing, while BLAKE2b-80 is clearly the worst performing algorithms of all. In this graphic we represented the interval of confidence for BLAKE2b-80 and SHA256-128 only due to the fact that all of the other series are very similar, and presenting all of the interval of confidence would make the graphic unreadable.

Analysing the Table 5.7, we realize that all of the algorithms performed with very similar execu-

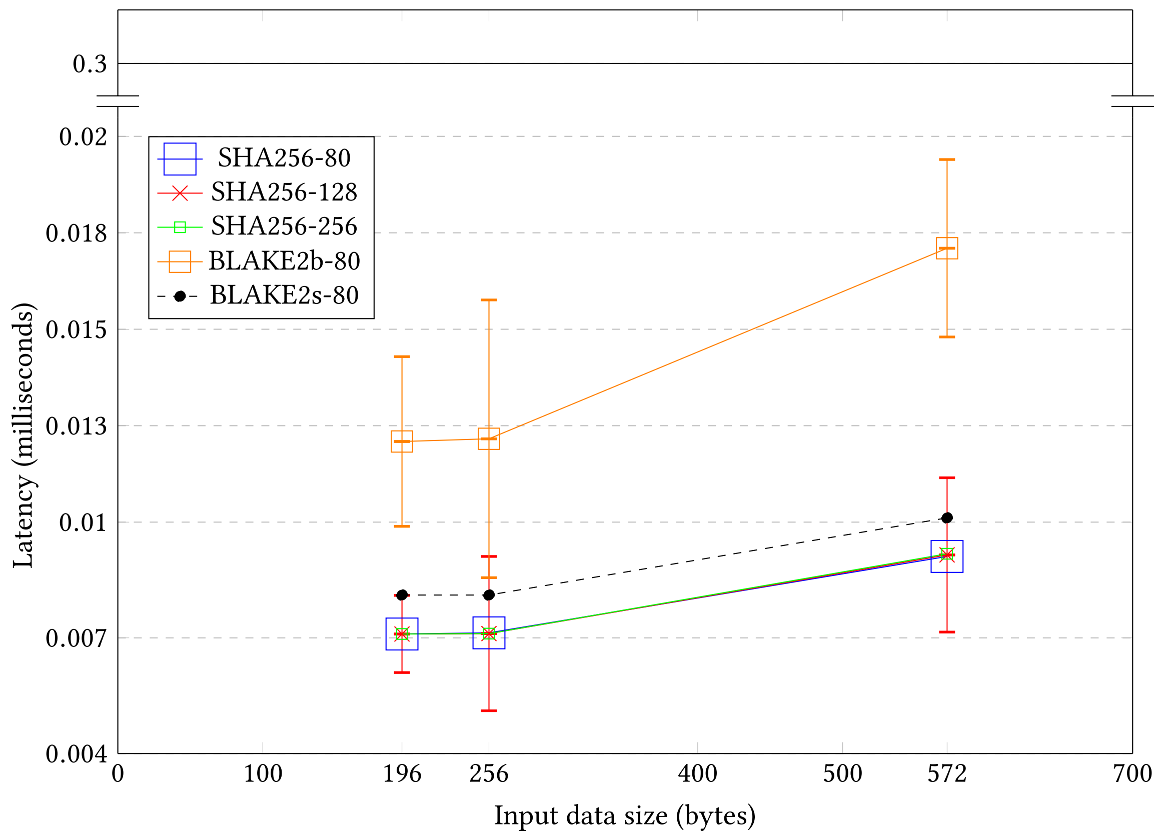


Figure 5.8: Progression of latency with input size in HMAC-based functions

Table 5.7: GMAC Functions performance analysis, using medium size data input

GMAC Algorithm	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
GMAC-AES128-64	0.005	0.001	0.167	0.005	± 0.001
GMAC-AES128-128	0.004	0.007	3.438	0.004	± 0.010
GMAC-AES256-64	0.005	0.001	0.218	0.005	± 0.001
GMAC-AES256-128	0.004	0.002	0.311	0.006	± 0.002

tion times. Once again, it was expected, as the only difference is the size of the generated MAC. In this case, we can see that the variants whose output is 64 bytes long, have a slightly higher execution time. It happens because in this variants we needed to reduce the default size of the MAC, doing an extra *memcpy* operation. Analysing the confidence intervals, we can see that they are all very similar for all input sizes. When using GMAC-AES128-128 with medium input size, we calculated an higher confidence interval (0.01 ms while all other are between 0.001 and 0.002). At the same time, we can see that the maximum value is also higher, reaching 3.4ms. Again, this behavior can be explained by the presence of outliers. As these outliers are only present in one of the experiments of one algorithm, the probable cause is that other process started consuming computing resources while the test was being performed.

Finally, as it happened on HMAC algorithms, here the maximum value is much higher than the average, but still below the targeted 0.3ms.

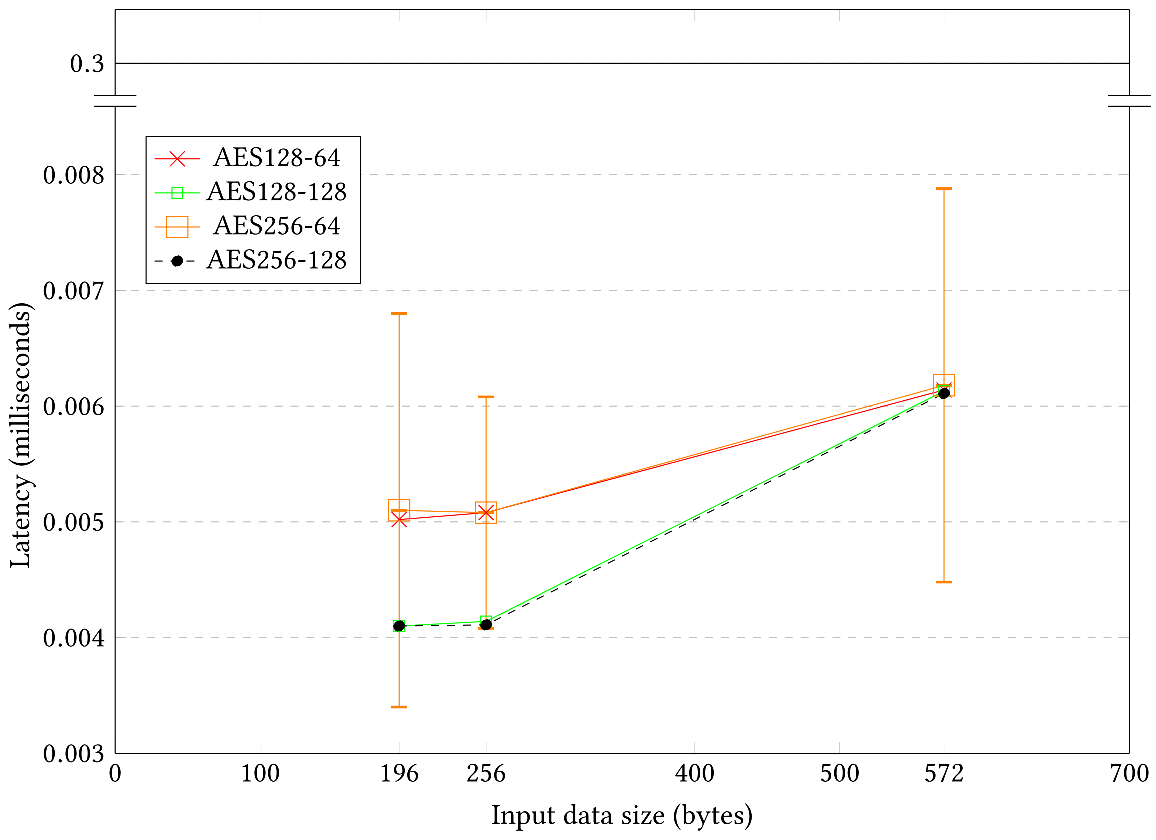


Figure 5.9: Progression of latency with input size in GMAC-based functions

From Figure 5.9, we can see that the algorithms performed very similarly in all input data sizes, with very small variations, specially when compared with HMAC algorithms. From this, we can see that these algorithms are consistent when changing the input data size. In this graphic we present the confidence interval of the AES256-64 algorithm. We should note that from the graphic, this algorithm has a slightly better performance with an input size of 256 bytes than with 196 bytes. This behavior can be justified with the confidence interval. We can see that for an input size of 196 bytes, the confidence interval is higher than for an input of 256 bytes.

Finally, comparing HMAC algorithms with GMAC algorithms, we can clearly see that GMAC has a better performance and are more consistent, either inside the same test (analysing the standard deviations and confidence interval), either comparing the latency difference when using different data sizes.

For encryption, we used different input data sizes, being 51 bytes, 204 bytes and 408 bytes. We did this because, for the same R-GOOSE packet, the amount of data used to generate the HMAC is not the same amount of data that is encrypted. On Table 5.8, we present the latency measurements for encryption and decryption algorithms.

From Table 5.8, the most important analysis we have to do is evaluating if this results show, or not, if encryption is a possibility in Power Distribution Systems. Analysing the measurements, we can say that having such algorithms running on a Raspberry Pi, make encryption feasible, as the additional latency introduced by these algorithms is very low and under our limit of 0.3ms.

Analysing the graphic from Figure 5.10, we can note that these algorithms grow in similar ways, except for AES256-Encryption, from small data sizes to intermediate. Again, this behavior can

Table 5.8: Encryption/Decryption Function performance analysis, using medium size data input

Function	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
AES256-GCM-Encrypt	0.005	0.002	0.205	0.005	± 0.002
AES128-GCM-Encrypt	0.009	0.004	0.337	0.008	± 0.005
AES256-GCM-Decrypt	0.010	0.003	0.260	0.009	± 0.005
AES128-GCM-Decrypt	0.008	0.003	0.271	0.007	± 0.005

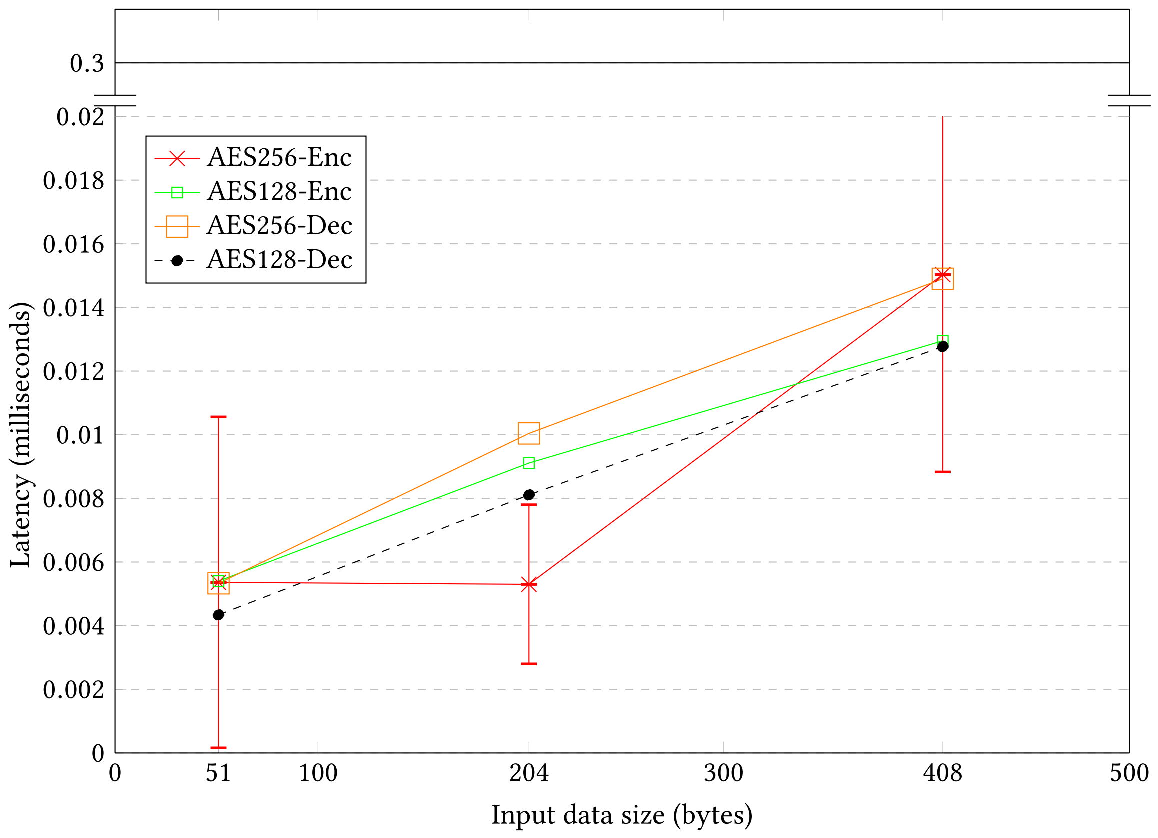


Figure 5.10: Progression of latency with input size in AES-based functions

explained looking at the confidence interval. In this case it is not presented on the graphic but for the input size of 51 bytes it is 0.0052ms and for the input size of 204 bytes is of 0.0025. This difference could derive from a set of outliers.

In an overall perspective, we can say that our library met our requirements in terms of performance. For the HMAC generation, we can clearly distinguish SHA256 variants with best performance, and BLAKE2b with much worse performance. For GMAC generation, all of the algorithms had similar performances, and all complied with the requirement. Finally, from the evaluation of AES-based functions, we conclude that encryption may be a viable solution on Power Distribution Systems, at least when used on the Raspberry Pi, more precisely, on our Security Gateway.

5.3.2 Protocol Related Functions

In this section, we will present the performance evaluation of the Protocol Related Functions. Similar to the method used to evaluate the cryptographic functions, we supplied an R-GOOSE packet to each function and we measured the time, in milliseconds, it took to execute. Again, we used the monotonic clock of the computer to measure that time.

As we did to analyse latency in cryptographic functions, we repeated each test 500 000 times, in order to obtain more precise values. We generated the R-GOOSE packets with our modified version of libIEC61850, and we used three different size packets. Our first packet is an R-GOOSE packet containing 1 GOOSE object, representing the minimum number of GOOSE objects that a packet could contain. The second packet contains 20 GOOSE objects and the last packet contains 220 GOOSE objects, representing the maximum number of GOOSE objects that an R-GOOSE packet could contain.

We performed the evaluation on every Protocol Related Functions variations, in other words, we analysed all combinations of Cryptographic functions for every Protocol Relation function. The following tables present the latency measurements for the R-GOOSE packet containing 20 GOOSE objects. Each table presents the results for one Protocol Related Function, where each entry indicates the cryptographic algorithm used.

Table 5.9: InsertHMAC performance analysis, using medium size data input (20 GOOSE objects)

HMAC Algorithm	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
HMAC-SHA256-80	0.011	0.010	1.018	0.01	± 0.014
HMAC-SHA256-128	0.010	0.011	1.057	0.01	± 0.015
HMAC-SHA256-256	0.011	0.009	0.932	0.01	± 0.013
HMAC-BLAKE2b-80	0.018	0.012	1.009	0.017	± 0.016
HMAC-BLAKE2s-80	0.011	0.011	0.951	0.011	± 0.015

Analysing the Table 5.9 we can verify that the latency measured executing the HMAC insertion on R-GOOSE packets isn't much higher than the execution time of unitary functions. This table is important because we will use this results to decide what cryptographic functions will be used when testing the Security Gateway. Following the same pattern as the unitary functions, we can identify the HMAC-BLAKE2b-80 as the worst performing function, being signalled in red. HMAC-SHA256-128, highlight in green, had the best performance for this tests, and will be used, as well as HMAC-BLAKE2b-80 to evaluate the Security Gateway. Analysing the confidence intervals of these algorithms, we can state that all of them will comply with our 0.3ms requirement.

In the Figure 5.11, is illustrated the evolution of the execution time with the number of GOOSE objects for InsertHMAC function. From the graphic, we can analyse that HMAC-SHA256-128 (in red), performed much better in all of the input packet sizes. On the other hand, BLAKE2b-80 is the worst performing algorithm, following the same pattern from the evaluation of the cryptographic functions.

On Table 5.10, we present the measurements for the *ValidateHMAC* function with all of the cryptographic algorithms, using the intermediate size R-GOOSE packet. As it was expected, the HMAC validation is faster than insertion, due to the generated HMAC not being copied to the original packet. Although, on the validation, we can see that the best performing algorithm is different from insertion. From the table it is clear that HMAC-BLAKE2b continues to have the

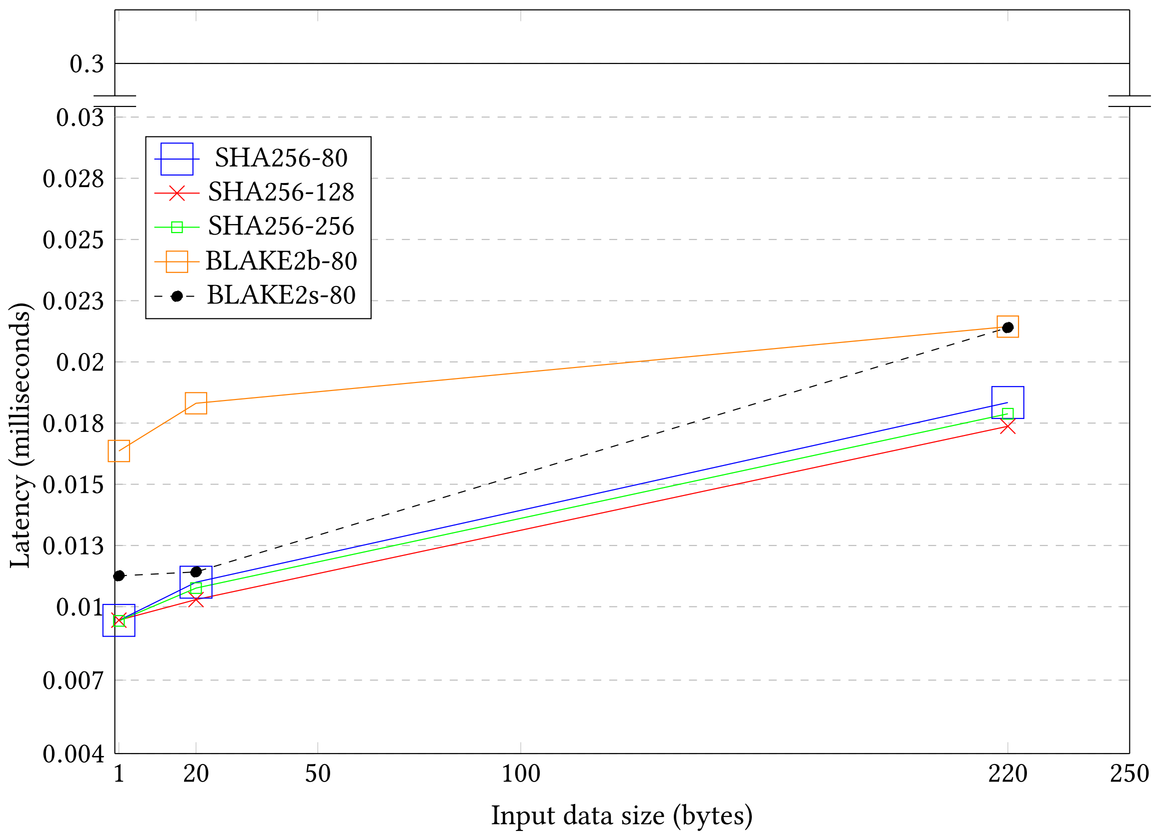


Figure 5.11: Progression of latency with number of GOOSE Objects in InsertHMAC Function

Table 5.10: ValidateHMAC performance analysis, using medium size data input

HMAC Algorithm	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
HMAC-SHA256-80	0.008	0.008	0.712	0.008	± 0.011
HMAC-SHA256-128	0.008	0.007	0.915	0.008	± 0.010
HMAC-SHA256-256	0.008	0.008	1.145	0.008	± 0.011
HMAC-BLAKE2b-80	0.014	0.008	0.59	0.014	± 0.011
HMAC-BLAKE2s-80	0.009	0.008	0.689	0.009	± 0.011

worst performance, having much higher execution time than HMAC-SHA-based algorithms. Although, when validating HMAC, we can see that HMAC-BLAKE2s variant is already very similar to SHA-based algorithms. Analysing the standard deviation and confidence interval, we can see that even in the worst case scenario, all of the algorithms still comply with the requirements we established.

In the Figure 5.12, we present the evolution of function performance when changing the size of the input packet. Again, from the graphic it is clear the difference between BLAKE2b-80 and the other algorithms in all of the input packet sizes. The SHA256 variants had very similar performances, namely SHA256-128 and SHA256-256. Because of that and the high confidence intervals that both of the functions have, we cannot say exactly which one is the best performing algorithm. Although, we choose SHA256-80 (green in the table) to be used on the protocol specific

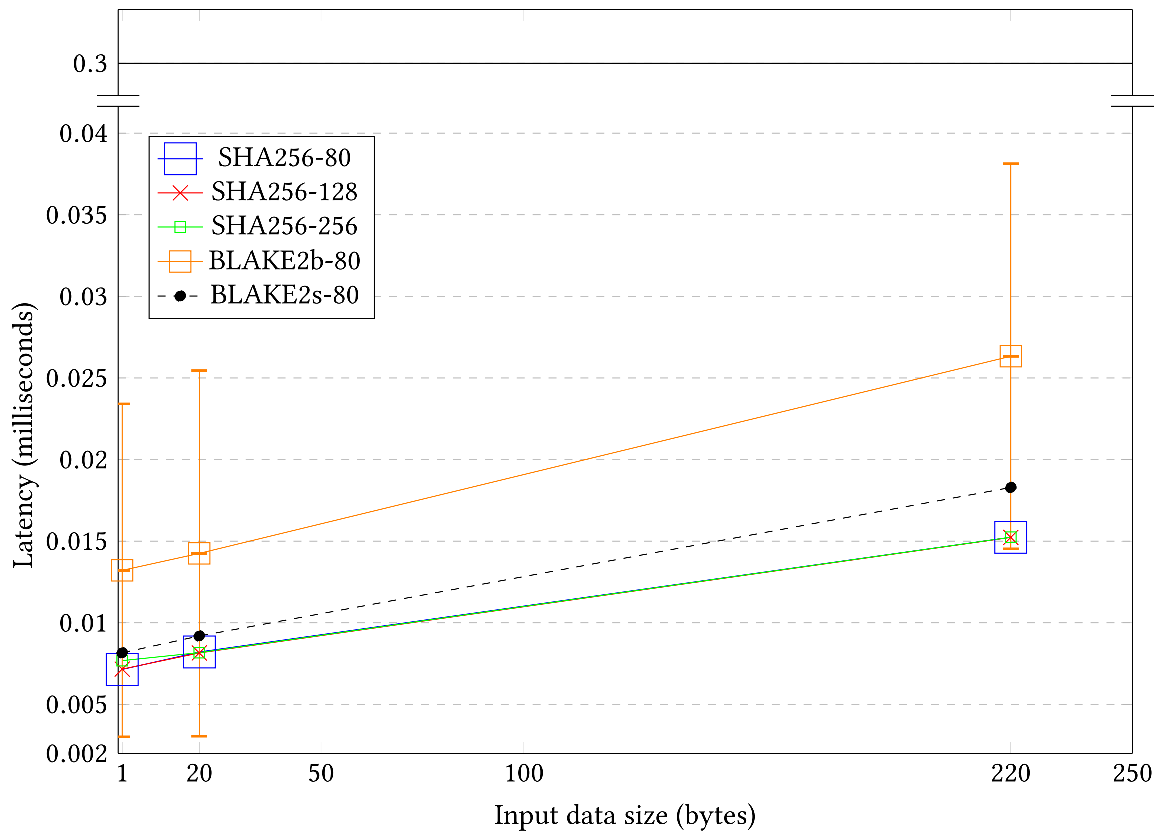


Figure 5.12: Progression of latency with number of GOOSE Objects in ValidateHMAC Function

evaluation. The reason behind this choice is that, when analysing the exact values, Table A.6 it has a slightly better performance in intermediate and large packet sizes. As for the worst performing algorithm, we choose BLAKE2b-80 (red in the table), due to the big difference in all of the packet sizes. We also present the confidence interval of BLAKE2b-80. Having such interval in consideration, we can state that even the worst performing algorithms complies with our 0.3ms requirement.

Table 5.11: InsertGMAC Functions performance analysis, using medium size data input

GMAC Algorithm	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
GMAC-AES128-64	0.007	0.009	1.027	0.007	± 0.012
GMAC-AES128-128	0.007	0.010	3.945	0.007	± 0.014
GMAC-AES256-64	0.008	0.010	2.821	0.007	± 0.014
GMAC-AES256-128	0.007	0.009	1.075	0.007	± 0.012

On the Table 5.11 we present the results from evaluation on InsertGMAC function using intermediate size packet. Analysing such table and the graphic on Figure 5.13, we can state that all of the algorithms have very similar behavior. This was expected as the operations performed in *Insert-GMAC* are the same for all of the algorithms, and, as we can see in Table 5.7, the cryptographic functions had very similar performances. All of the algorithms had very similar performances and as the confidence intervals are high, we cannot say exactly which one had the best perfor-

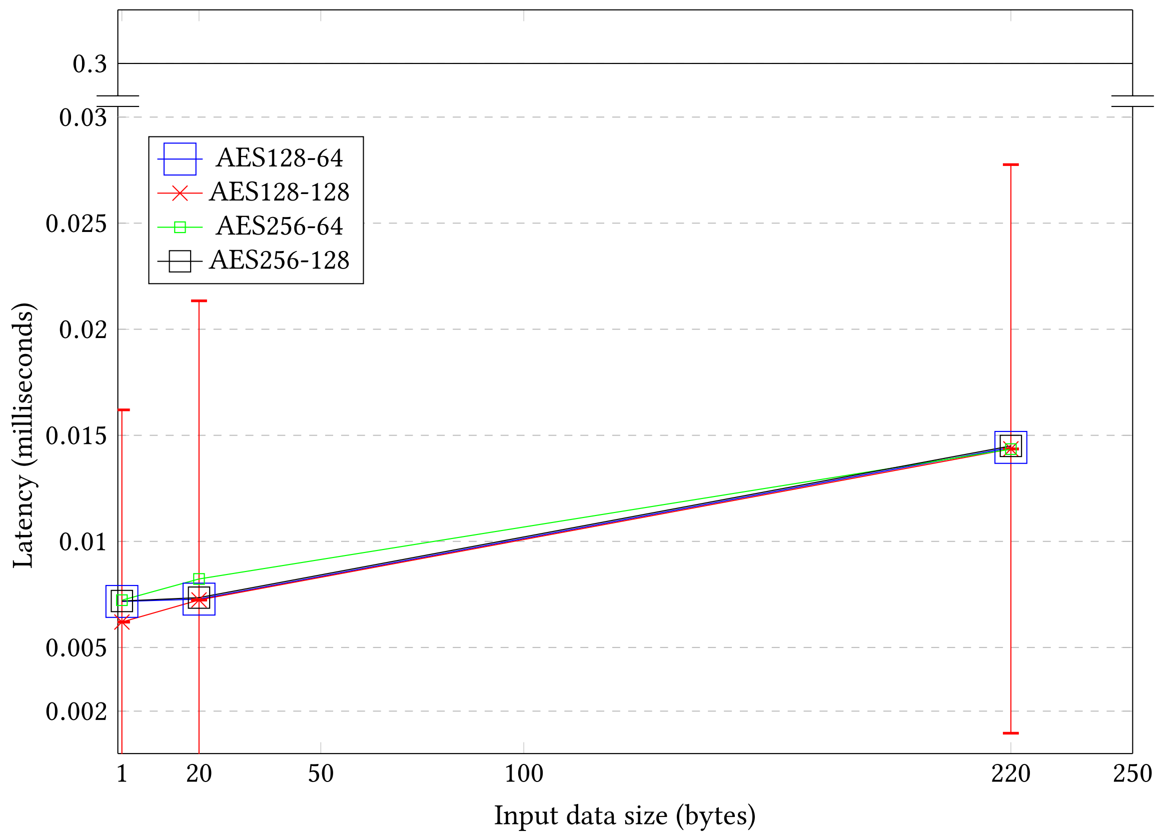


Figure 5.13: Progression of latency with number of GOOSE Objects size in InsertGMAC function

mance. Although, analysing the complete testing tables, we can see that has a slightly better performance in both small and intermediate packet sizes. As for the worst performing algorithm we choose AES256-128, having the worst performances in small and large packet sizes. The confidence intervals are similar for all of the algorithms except for AES256-128 using small input sizes. In this case, the confidence interval is 0.02 ms. We can also verify the existence of outliers, as for example, the maximum value of 5.871 ms. Yet, all of the algorithms comply with the 0.3ms requirement.

Table 5.12: ValidateGMAC Functions performance analysis, using medium size data input

GMAC Algorithm	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
GMAC-AES128-64	0.006	0.007	0.0774	0.005	± 0.001
GMAC-AES128-128	0.005	0.006	0.852	0.005	± 0.009
GMAC-AES256-64	0.006	0.006	0.679	0.006	± 0.009
GMAC-AES256-128	0.005	0.006	0.704	0.005	± 0.009

On the Table 5.12 we present the results from ValidateGMAC evaluation using intermediate packet size. We can see that all of the algorithms performed with very similar latency times, with latency oscillating between 0.005 and 0.006 milliseconds. As we stated before, the GMAC generation functions had very similar performances, making this outcome expected. We must note that, having in consideration the measurements for the average and the confidence interval, all of this algorithms comply with our performance requirement.

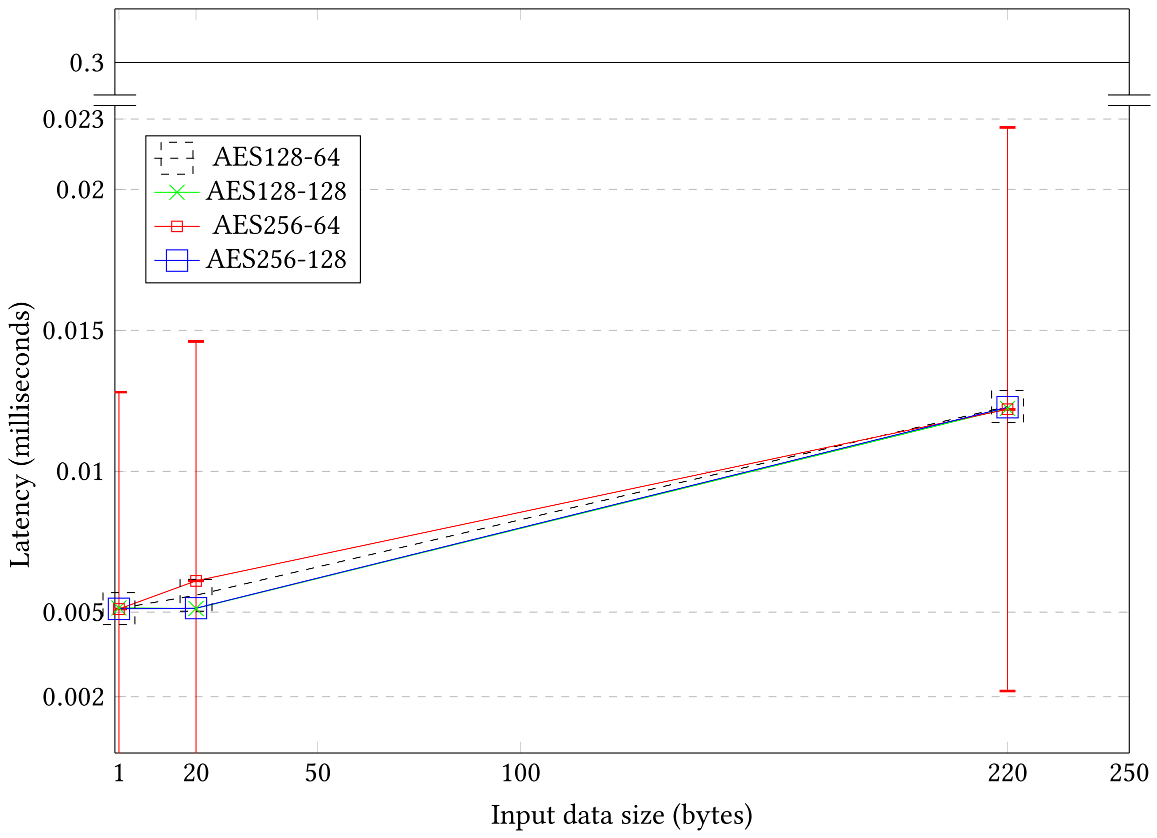


Figure 5.14: Progression of latency with number of GOOSE Objects size in ValidateGMAC function

Analysing the graphic on Figure 5.14, we realize that the variation of the latency with the size of the input is small. For example, analysing AES128-128 in more detail, this algorithm took 0.005 ms to execute with intermediate packet size, and 0.012 with the large packet size. Having in consideration that the large packet size is eleven times larger than the intermediate packet, the increment on the latency can be considered reduced. Again, all of these algorithms had very similar performances and high confidence intervals. Given that, we cannot say exactly which one is the best performing algorithm. Although, analysing the complete testing tables, we can see that AES128-128 had the best performances for small and large packet sizes. On the other hand, AES256-64 was considered the worst performing algorithm, having the worst performances for small and intermediate packet sizes.

Table 5.13: Encryption Function performance analysis, using medium size data input

Function	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
AES256-GCM-Encrypt	0.013	0.011	3.941	0.013	± 0.015
AES128-GCM-Encrypt	0.011	0.009	0.797	0.011	± 0.012
AES256-GCM-Decrypt	0.011	0.007	0.677	0.011	± 0.009
AES128-GCM-Decrypt	0.011	0.007	0.870	0.011	± 0.009

The last table (5.13) of this section, presents the evaluation performed on Encryption and Decryption functions. They are presented together and, because of that, we selected two best performing algorithms and two worst performing algorithms from the same table. Analysing the test case presented, we can note that encryption could be a feasible mechanism to be applied on Power Distribution Systems. Comparing each algorithm, we can see that AES with 128 bits key length has a better performance. Another observation we must do is that the decryption functions have better performance measurements against the encryption function, being a common behavior in cryptography.

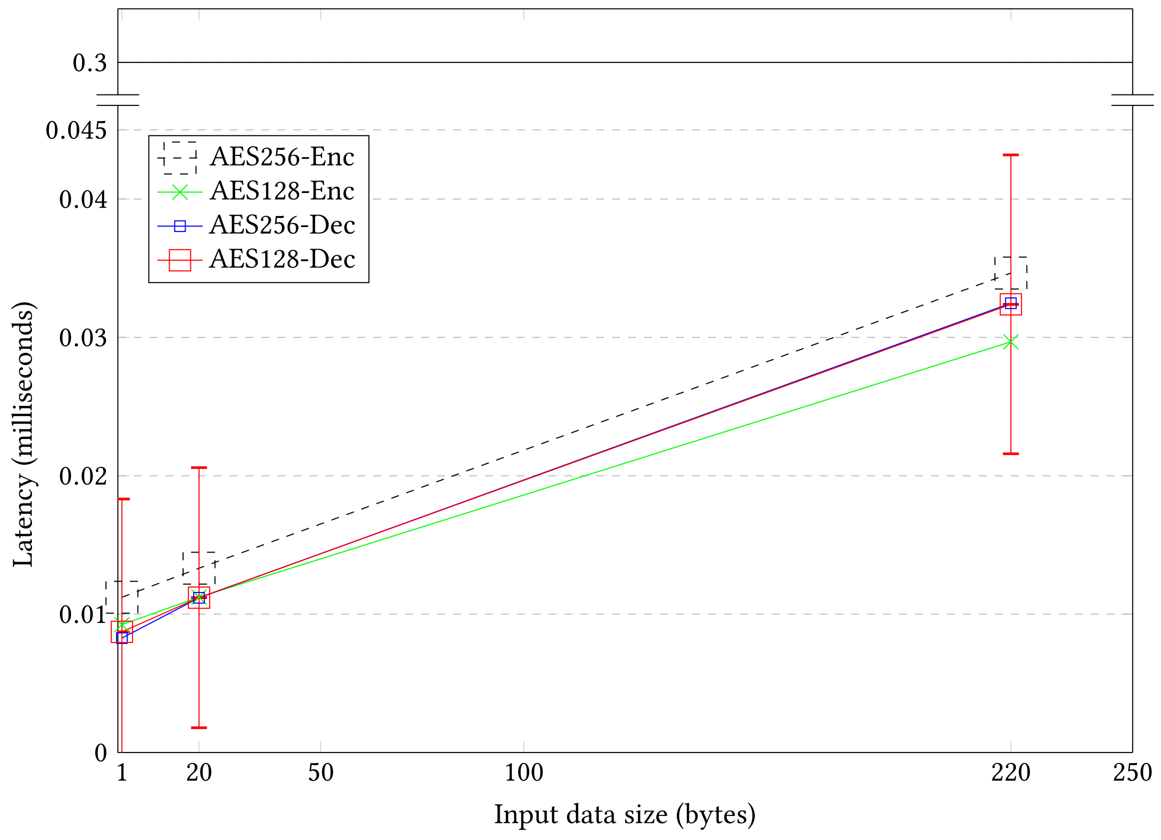


Figure 5.15: Progression of latency with number of GOOSE Objects in Encrypt And Decrypt Functions

From Figure 5.15, we note a downside on encryption and decryption. The execution time for encryption and decryption grows much faster than execution time for HMAC/GMAC insertion and validation. This was also expected, due to the inner properties and differences between encryption and hashing. This values still show that encryption is a feasible solution when used on the Raspberry Pi, however this fast growth may represent a serious problem when encryption is applied in IEDs. From the graphic and the table, we choose AES128 variants as the best performing algorithms, while AES256 variants are the worst performing algorithms. We should also note that AES256-Dec and AES128-Dec have very similar behaviors and, due to that, they are overlapping in the graphic. Also, analysing the confidence intervals of these algorithms we can state that they comply with our 0.3ms requirement for all input data sizes.

After fully testing the Security Library, we state that all of the functions met the requirement of executing in less than 0.3 milliseconds. In terms of HMAC insertion and validation, the SHA256 variants performed significantly better than BLAKE2b variants, while BLAKE2s got a similar performance to SHA256. For the GMAC insertion and validation, there were not significant differences between the algorithms tested. We can also state that GMAC had a better performance than HMAC, with lower latency measurements in insertion and in validation. For encryption, we should note two aspects: with these tests we showed that is possible to comply with the performance requirements when applying encryption on the Raspberry Pi, but at the same time, we realized that the execution time to encrypt an R-GOOSE packet grows considerably fast when incrementing the packet size, what could compromise the performance in devices with limited capabilities.

5.3.3 Raspberry Pi Bridging Device

In this section, we will present the performance evaluation of the Raspberry Pi acting as a bridging device. In this stack of tests, we analysed only the bridging component and the impact it has on overall communications, not being focused on R-GOOSE.

We measured and analysed two metrics to evaluate the performance of the bridging device: bandwidth and latency of communications. To measure the bandwidth, we used iPerf3 [1] tool. To measure the latency, we used Ping tool, configuring its arguments to modify the test conditions. In these experiments, we used the setup illustrated in Figure (5.1).

Bandwidth Test

iPerf3 [1] is a tool used to measure the quality of communications in an IP network. This tool allows to measure the maximum bandwidth achievable, the loss and other metrics about the network. For our tests, we will be focused on measuring the bandwidth of communications, as it will be one of the metrics that will have more impact on the Security Gateway performance

iPerf3 measures the bandwidth using a client and a server that must be installed in two endpoints of the network. It establishes a connection between them and tries to transfer as many data as it is possible during a given amount of time. With those values, it calculates the bandwidth of that connection. We installed iPerf Server on PC A and the iPerf Client on PC B and we ran the experiment during 10 minutes. The results were the following:

Table 5.14: Total data transferred and Bandwidth measurements with bridge device in place

Total data transferred	6.62 GBytes
Bandwidth	94.7 Mbits/sec

We then repeated the same test but without the Raspberry Pi, connecting both PCs directly and we measured the following results:

Table 5.15: Total data transferred and Bandwidth measurements without bridge device in place

Total data transferred	6.63 GBytes
Bandwidth	94.9 Mbits/sec

Comparing Table 5.14 and Table 5.15 we can see that bandwidth decreased from 94.9 Mbits/sec, when Raspberry Pi is not in place, to 94.7 Mbits/sec when it is in place. We can conclude that our bridging device does not decrease significantly the bandwidth of the network used.

Latency Test

To measure latency that our bridging device introduces on the communications we used the well known Ping tool. We used the setup as it is shown in the Figure 5.16 and we executed the Ping command from PC B.

The Ping tool measures the RTT of an ICMP packet in a given network. In terms of latency, our final goal is to analyse if our Security Gateway can apply the security mechanisms developed without compromising the 3ms of latency per R-GOOSE packet. This 3ms are the Transfer Time, as illustrated in the Figure 3.4. There we can see that it is an OTT (One Trip Time) and does not include the latency introduced by the endpoint applications (R-GOOSE Applications).

Given that and the fact that Ping measures the RTT, we can define our objective for this test as achieving an RTT lesser to 6ms. If we assume that both trip times on Ping calculation are very similar (as they use the same network and packet size are very similar), we can assume that Transfer Time will be lesser than half of the RTT calculated by Ping, because Transfer Time does not include the processing time on each application.

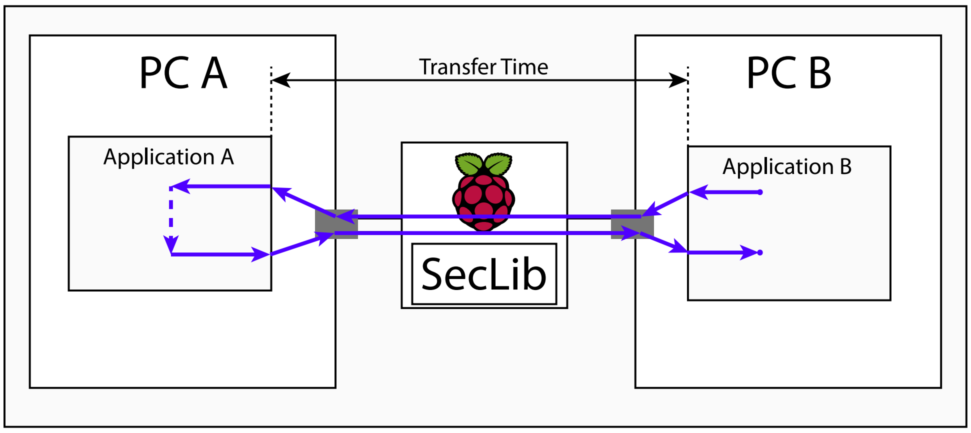


Figure 5.16: Diagram of Ping testing and relation with Transfer Time

For this test, we used a set of 12000 ICMP packets to measure the average latency of each packet. Also, we repeated the test several times, changing the throughput rate of the ping command. From R-GOOSE specifications, we can see that when an event is raised, a message is sent every 4ms at first, then incrementing this time interval. With this in mind, we analysed the latency of the bridge, sending packets with a time interval of 4ms, 10ms, 20ms, 30ms, 40ms, 50ms and 1 second.

Analysing the Table 5.16, we can confirm that our bridging device complies with the requirements for the communications, more precisely, with the 3ms for Transfer Time. The averages for RTT of ICMP packets are between 2.548ms and 3.242ms. From there we can estimate an OTT of 1.274ms to 1.621ms. Moreover, if we take in consideration the standard deviation the latency measurements will still comply with the requirements. On the following graphic is plotted the latency related with the time interval between packets. From the graphic, we can see that in the case test simulating the packet rate after an R-GOOSE event been raised, we achieved the best performance, increasing the latency with the time interval between packets. We can also note that the RTT tends to a value closer of 2.8ms, as we can see for the test with 1 second of time interval between packets.

From the analysis done in this subsection, we can conclude that our bridging device complies with

Table 5.16: Bridge device latency measurements

Time Interval	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)
4	2.548	0.566	13.361	1.736
10	2.611	0.515	10.501	1.890
20	2.652	0.411	8.579	1.957
30	2.669	0.490	9.081	1.694
40	2.719	0.547	9.135	1.970
50	2.771	0.515	9.464	1.880
1000	2.800	0.426	11.819	2.066

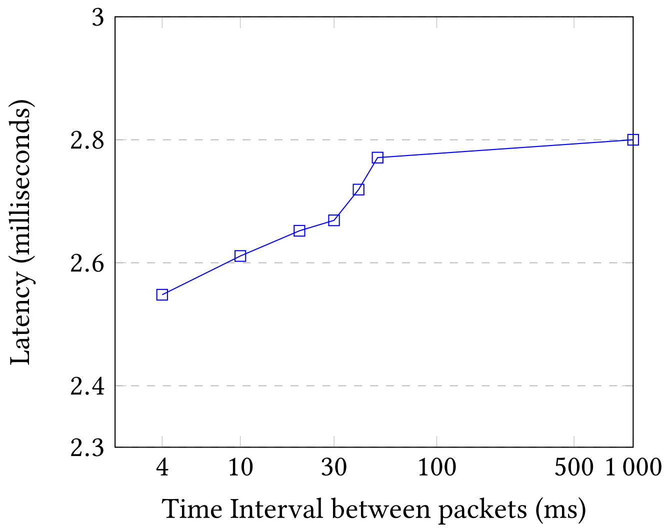


Figure 5.17: Progression of latency with the rate of packets sent using Ping

the performance requirements for the communications in Power Distribution Systems. We estimate that the OTT of the communications passing through the Raspberry Pi is between 1.274ms and 1.621ms, leaving us much space to apply the security mechanisms without compromising the 3ms requirement.

5.3.4 Security Gateway

In this section we will evaluate the performance of our Security Gateway, that is, our Raspberry Pi Bridging Device applying the security mechanisms to R-GOOSE packets. In these tests we were focused on analysing the latency on R-GOOSE communications passing through the Security Gateway. To measure latency, we developed an application that was acting as an UDP/R-GOOSE Server. This application was placed in PC A and was receiving R-GOOSE packets sent from PC B. On PC B, packets were generated and sent using our modified version of libIEC61850 [42].

We repeated each test 1000 times and we will present the measurements when the packets were sent with a time interval of 4ms. The reason we choose this time interval is because it is the packet rate used by R-GOOSE applications when events are raised. We tested all of the protocol related functions of Security Library using two cryptographic algorithms selected in the Section 5.3.2. As explained before, we choose the best and the worst performing algorithm.

As we did in the analysis of Ping results for bridging device, in these tests we are also measuring

Table 5.17: Bridge device latency (RTT) measurements

F. Name	Average (ms)	Confidence Interval 95%	Maximum (ms)	Minimum (ms)	Result
AddHMAC-SHA256-128	3.165	0.749	8.126	1.829	Pass
AddHMAC-BLAKE2b-80	3.122	0.735	7.687	2.315	Pass
AddGMAC-AES128-128	3.100	0.807	12.148	2.266	Pass
AddGMAC-AES256-128	3.121	0.781	7.011	2.231	Pass
ValidateHMAC-SHA256-80	3.155	1.016	10.75	2.289	Pass
ValidateHMAC-BLAKE2b-80	3.073	1.014	11.234	2.273	Pass
ValidateGMAC-AES128-128	3.184	0.964	11.205	2.28	Pass
ValidateGMAC-AES256-64	3.051	1.000	10.067	2.237	Pass
Encrypt-AES128	3.069	0.829	8,688	2.238	Pass
Encrypt-AES256	3.207	0.855	12.689	2.41	Pass
Decrypt-AES128	3.129	1.050	11.661	2.162	Pass
Decrypt-AES256	3.046	0.755	12.989	2.243	Pass

the RTT and not the Transfer Time. Although, as we showed on Figure 5.16, we can estimate the OTT (that is always bigger than Transfer Time of Figure 3.4) and evaluate if the test was successful or not. From the Table 5.17 we can see that all of the RTT values are between 3.046 ms and 3.207 ms, from where we can estimate an OTT between 1.523 ms and 1.6035 ms. With this OTT values, we can state that our Security Gateway complies with the R-GOOSE requirement of a Transfer Time minor than 3ms.

Concluding this chapter, we tested each component that was developed. For the Security Library, we tested the functional requirements of each functions, using standardized test vectors for cryptographic functions, and R-GOOSE packets for protocol related functions. All of the tests performed were successful. In terms of performance, we analysed all of the functions individually. Our objective was to understand the latency that each function would introduce when applied directly on another device, as an IED. All of these tests were successful. Finally, we analysed the performance of the bridge and the Security Gateway (Security Library combined with Raspberry Pi Bridging Device). We evaluated the impact that the bridge would have in the communications bandwidth, decreasing from 94.9 Mbits/sec to 94.7 Mbits/sec. Then we analysed the latency of R-GOOSE communications and compared with the requirements from Transfer Time. We estimated that the OTT on our scenario would be between 1.523 ms and 1.6035 ms. With these values, we can conclude that our solution complies with the performance requirements.

Chapter 6

Conclusions and Future Work

The objective of this research work was to develop and validate a set of security mechanisms for Critical Systems, to ensure a new set of security requirements acquired with the shift to 5G networks. In order to achieve such goal, we analysed three Critical Systems: Railway Signalling Systems, Power Distribution Systems and PPDR Platforms. From those systems, we defined a set of security and performance requirements that our solution must comply with.

During this work we faced some challenges. Initially, it was very difficult to find information on the protocols used by each system. The majority of such protocols are proprietary, and there is few public information detailing its specifications. Moreover, was extremely difficult to find open implementations of such protocols. Fortunately, we found an open implementation of GOOSE protocol, that later we modified to generate R-GOOSE packets.

In terms of the analysis done of the protocols, the most relevant part is the requirements and the gap analysis performed. To set the requirements, we analysed the standards and recommendations that regulate each area. Then, in the gap analysis, we compared the requirements we collected with the security mechanisms that each protocol was already implementing, and we concluded that was necessary to ensure message integrity, message authentication and message confidentiality.

After the gap analysis, we had our objectives defined and, with that in mind, we proposed the development of a Common Framework for Critical Systems, composed by a Security Library and Bridging Device, that combined would provide a Security Gateway for Critical System's devices. The Security Library developed consists on a set of cryptographic functions that can be used in any area, being based on the OpenSSL library. Moreover, the library includes a set of protocol related functions that apply the security mechanisms to a specific protocol. In this research work, due to the non existence of other open implementations that we had access to, the protocol related part was only focused on Power Distribution Systems, more specifically on the R-GOOSE protocol, although other protocols can be supported. We developed functions to ensure Integrity, Authentication and Confidentiality of R-GOOSE messages.

For the Bridge Device, we tried several methods to perform an efficient bridging, using a Raspberry Pi as hardware. Our final solution consists on establishing a Linux Bridge (brctl) and using IPTables alongside with a Netfilter Kernel Module, to bridge packets and modifying them in a user space application.

We evaluated all of the developed components in terms of functionality and in terms of performance. For the functionality, we proved that our solution was able provide the security requirements necessary. In terms of performance, we evaluate the Security Library by itself, the bridging device and both combined, forming the Security Gateway. The final results showed that our Security Gateway was a feasible solution on the context of Power Distribution Systems. By the requirements of protocol R-GOOSE, each message should have a Transfer Time of at most, 3ms. Adding our Security Gateway to the system, we were able to achieve an estimated Transfer Time between 1.523 ms and 1.6035 ms, for integrity/authentication and encryption (when applied in separate). Furthermore, from the analysis made on the Security Library by itself, we showed that, on a device with similar capabilities of a Raspberry Pi 4B, all of the security mechanisms could be applied without breaking the performance requirements.

As this research work is part of 5G Mobilizer project, our solution will be integrated in a bigger and more realistic testbed, that will allow us to understand the impact that our solution will have in a much more realistic network, as well as analyse the performance of our Security Library when applied directly in IEDs. Also, as other future work, our Security Library could be expanded to support other protocols used on Critical Systems. In this version of our library, it already implements the mechanisms to provide authentication, although those mechanisms are dependent of a Key Management System that is still not included on the library. The development of such Key Management System should be considered in a future work. Finally, as there are several legacy devices deployed on real environments that cannot even support R-GOOSE, arises the opportunity of adding a module to our Security Gateway, capable of converting unsecured GOOSE packets on secured R-GOOSE packets.

References

[1] " iPerf3 ". https://iperf.fr.

[2] 5G Mobilizer Project Consortium. " 5G Mobilizer Project ". Available: https://5go.pt/.

[3] ALSETLab. " Khorjin - An IEC 61850-90-5 Gateway for Synchrophasor Data Transfer with support for IEEE C37.118.2 ". github.com/ALSETLab/Khorjin-IEC61850-90-5.

[4] American Public Transportation Association. " Securing Control and Communications Systems in Rail Transit Environments ". Tech Rep APTA SS-CC-03-15, April 2015. [Online]. Available: https://www.apta.com/wp-content/uploads/Standards_Documents/APTA-SS-CC-03-15.pdf.

[5] A. Apostolov. R-GOOSE: what it is and its application in distribution automation. *CIRED - Open Access Proceedings Journal*, 2017(1):1438–1441, 2017.

[6] Jean-Philippe Aumasson and Markku-Juhani O. Saarinen. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693, RFC Editor, November 2015.

[7] Pavel F. Baranov, Sergey V. Muravyov, Almaz O. Sulaymanov, and Lyudmila I. Khudonogova. Software for Emulating the Sampled Values Transmission in Accordance with IEC 61850 Standard. *2nd International Symposium on Computer, Communication, Control and Automation (3CA 2013)*, 2013.

[8] Daniel J. Bernstein. The Poly1305-AES message-authentication code. 2005.

[9] Daniel J. Bernstein. The Salsa20 family of stream ciphers. 2007.

[10] H. Birkholz, C. Krauß, M. Zhdanova, D. Kuzhiyelil, T. Arul, M. Heinrich, S. Katzenbeisser, N. Suri, T. Vateva-Gurova, and C. Schlehuber. A Reference Architecture for Integrating Safety and Security Applications on Railway Command and Control Systems: Extended Abstract. In *Proceedings of 4th International Workshop on MILS: Architecture and Assurance for Secure Systems (MILS'18)*, 2018.

[11] Kwasi Boakye-Boateng and Arash Habibi Lashkari. Securing GOOSE: The Return of One-Time Pads. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8, 10 2019.

[12] D. Saraiva, D. Corujo and R. Aguiar. IEC 61850 Data Transfer Evaluation over Public Networks. *Wireless Personal Multimedia Communications Symp. - WPMC*, 2019.

[13] Mauricio Gadelha da Silveira. IEC 61850 Network Cybersecurity : Mitigating GOOSE Message Vulnerabilities. In *6th Annual PAC World Americas Conference*, Raleigh, North Carolina, August 2019.

[14] M. Daoud and X. Fernando. On the Communication Requirements for the Smart Grid. *Energy and Power Engineering*, 3, January 2011.

[15] Diogo Saraiva. " LibIEC61850-R-GOOSE ". https://github.com/DiMSaraiva/LibIEC61850-R-GOOSE.

[16] Eduardo Andrade. " R-GOOSE-SecLib ". https://github.com/slipz/R-GOOSE_SecLib.

[17] Eduardo Andrade. " RPi-Gateway ". https://github.com/slipz/RPi-Gateway.

[18] Ahmed Elgargouri, Reino Virrankoski, and Mohammed Elmusrati. IEC 61850 Based Smart Grid Security. *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, 2015, 03 2015.

[19] European Committee for Electrotechnical Standardization (CENELEC). "Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems". Standard EN 50159, Setembro 2010.

[20] European Committee for Electrotechnical Standardization (CENELEC). "Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling". Standard EN 50129, Novembro 2018.

[21] Shaik Mullapathi Farooq, S. M. Suhail Hussain, and Taha Selim Ustun. Performance Evaluation and Analysis of IEC 62351-6 Probabilistic Signature Scheme for Securing GOOSE Messages. *IEEE Access*, March 2019.

[22] Seyed Reza Firouzi, Luigi Vanfretti, Albert Ruiz-Alvarez, Hossein Hooshyar, and Farhan Mahmood. Interpreting and implementing IEC 61850-90-5 Routed-Sampled Value and Routed-GOOSE protocols for IEEE C37.118.2 compliant wide-area synchrophasor data transfer. *Electric Power Systems Research*, 144:255–267, 03 2017.

[23] Working Group CYSIS – Subgroup Security for Safety. Security for Safety. https://www.seceng.informatik.tu-darmstadt.de/media/seceng/ag_cysis/SD52018Kant.pdf, 2018.

[24] Shailendra Fuloria and Ross Anderson. The Protection of Substation Communications. In *SCADA Security Scientific Symposium*, 2010.

[25] Shailendra Fuloria, Ross Anderson, Kevin Mcgrath, Kai Hansen, and Fernando Alvarez. The protection of substation communications. 2010. Available: https://www.cl.cam.ac.uk/ rja14/Papers/S4-2010.pdf.

[26] A. Hadbah, A. Kalam, and A. Zayegh. Powerful IEDs, ethernet networks and their effects on IEC 61850-based electric power utilities security. In *2017 Australasian Universities Power Engineering Conference (AUPEC)*, pages 1–5, Nov 2017.

[27] M. Heinrich, T. Vateva-Gurova, T. Arul, S. Katzenbeisser, N. Suri, H. Birkholz, A. Fuchs, C. Krauß, M. Zhdanova, D. Kuzhiyelil, S. Tverdyshev, and C. Schlehuber. Security Requirements Engineering in Safety-Critical Railway Signalling Networks. *Security and Communication Networks*, July 2019.

[28] M. Heinrich, J. Vieten, T. Arul, and S. Katzenbeisser. Security Analysis of the RaSTA Safety Protocol. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, Miami, FL, USA, 2018.

[29] J. Hoyos, M. Dehus, and T. X. Brown. Exploiting the GOOSE protocol: A practical attack on cyber-infrastructure. In *2012 IEEE Globecom Workshops*, pages 1508–1513, Dec 2012.

[30] S. M. S. Hussain, S. M. Farooq, and T. S. Ustun. Analysis and Implementation of Message Authentication Code (MAC) Algorithms for GOOSE Message Security. *IEEE Access*, 7:80980–80984, 2019.

[31] International Electrotechnical Commission. " Power Utility Automation ". Standard IEC 61850.

[32] International Electrotechnical Commission. " Functional safety of electrical/electronic/programmable electronic safety-related systems ". Standard IEC 61508, 2010.

[33] International Electrotechnical Commission. " Power systems management and associated information exchange - data and communications security ". Standard IEC 62351, 2018.

[34] International Electrotechnical Commission. " Telecontrol equipment and systems - Part 5: Transmission protocols ". Standard IEC 60870, 2018.

[35] International Union of Railways. " ARGUS Project ", 2015. Available: https://www.uic.org/com/uic-e-news/392/article/argus-kick-off-meeting-paris-19?page=modal_enews.

[36] O. Khaled, A. Marín, F. Almenares, P. Arias, and D. Díaz. Analysis of secure TCP/IP profile in 61850 based substation automation system for smart grids. In *International Journal of Distributed Sensor Networks, p. 5793183*, 2016.

[37] Rafiullah Khan, Kieran Mclaughlin, David Laverty, and Sakir Sezer. Design and implementation of security gateway for synchrophasor based real-time control and monitoring in smart grid. *IEEE Access*, 5:11626 – 11644, 06 2017.

[38] Kamil Kołtyś and Robert Gajewski. SHaPe: A Honeypot for Electric Power Substation. *Journal of Telecommunications and Information Technology*, pages 37–43, January 2015.

[39] Nishchal Kush, Mark Branagan, Ernest Foo, and Ejaz Ahmed. Poisoned GOOSE : exploiting the GOOSE protocol. *Conferences in Research and Practice in Information Technology Series*, 149, 01 2014.

[40] I. Lopez and M. Aguado. Cyber Security Analysis of the European Train Control System. In *IEEE Communications Magazine, vol. 53, no. 10, pp. 110–116*, 2015.

[41] David A. McGrew and John Viega. The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH. RFC 4543, RFC Editor, May 2006.

[42] MZ Automation. " Official repository for libIEC61850, the open-source library for the IEC 61850 protocols ". github.com/mz-automation/libiec61850.

[43] Magnus Nystrom. Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512. RFC 4231, RFC Editor, December 2005.

[44] Sebastian Obermeier, Roman Schlegel, and Johannes Schneider. Assessing the Security of IEC 62351. *Proceedings of the 3rd International Symposium for ICS & SCADA Cyber Security Research 2015*, pages 11–19, 01 2015.

[45] OpenSSL. " OpenSSL 1.1.1 ". https://www.openssl.org/news/openssl-1.1.1-notes.html.

[46] Upeka Premaratne, Jagath Samarabandu, Tarlochan Sidhu, Robert Beresh, and Jian-Cheng Tan. Security Analysis and Auditing of IEC61850-Based Automated Substations. *Power Delivery, IEEE Transactions on*, 25:2346 – 2355, 11 2010.

[47] 5G Mobilizer Project. Deliverable D3.2 and D3.3 - Use cases, requirements and solutions architecture for M2M critical communications, 2019.

[48] G. Samta, S. Karmakar, M. Sharma, and S. Sharma. Bluetooth Secure Simple Pairing with enhanced security level. In *Journal of Information Security and Applications, vol. 44, pp. 170-183*, February 2019.

[49] C. Schlehuber, M. Heinrich, T. Vateva-Gurova, S. Katzenbeisser, and N. Suri. Challenges and Approaches in Securing Safety-Relevant Railway Signalling. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2017.

[50] J. Schuler and P. Favre-Perroz. IEC-61850, Inter-substation communication: Optimal signed-crypted R-GOOSE and R-Sampled Values on IP-Multicast networks. 06 2016.

[51] Dae-Yong Shin, Sugwon Hong, Il Lim, and Seung-Jae Lee. Evaluation of Security Algorithms for the SCADA System based on IEC 61850. *EmbeddedCom-ScalCom*, January 2009.

[52] J. Smith, S. Russell, , and M. Looi. Security as a safety issue in railcommunications. In Australian Computer Society, Inc., editor, *Proceedings of the 8th Australian workshop onSafety critical systems and software-Volume 33, pp. 79-88*, 2003.

[53] National Institute Of Standards and Technology. *NIST Special Publication 800-38D Computer Security - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.* NIST, Gaithersburg, MD 20899-8930, 2007.

[54] Swiss Re Institute. Natural catastrophes and man-made disasters in 2017: a year of record-breaking losses, 2018. Available: https://reliefweb.int/sites/reliefweb.int/files/resources/sigma1_2018_en.pdf.

[55] The SECRET Consortium. " Security of railways against eletromagnetic attacks ". UIC-ETF. ISBN: 978-2-7461-2465-3, November 2015. [Online]. Available: https://cordis.europa.eu/docs/results/285/285136/final1-white-paper-security-light.pdf.

[56] UNISIG. "RBC-RBC Safe Communication Interface". Subset-098, May 2007.

[57] The Railway Technical Website. Signalling. http://www.railway-technical.com/signalling/.

[58] Ye Yan, Yi Qian, Hamid Sharif, and David Tipper. A survey on cyber security for smart grid communications. *Communications Surveys and Tutorials, IEEE*, 14:998–1010, 01 2012.

This page is intentionally left blank.

Appendices

This page is intentionally left blank.

Appendices A

Appendix A

In this appendix we present the complete results from the performance testing of our Security Library. The testing methodology used is explained in detail in Section 5.3. In brief, we used a Raspberry Pi 4B to run each functions. In each test, we provide a given input with a given size (specified in each table entry), and we measured the execution time it took using Raspberry Pi's monotonic clock. Each experience was repeated 500 000 times.

Each table entry is specified by the algorithm we are analysing and the input data size. For each combination, we measure average in milliseconds, standard deviation in milliseconds, the maximum value, the minimum value and the confidence interval of 95%. A short version of each table and respective analysis is presented in Section 5.2.

On table A.1 we present the complete results for the HMAC cryptographic functions evaluation, followed by the Table A.2 presenting the evaluation of GMAC cryptographic functions. On Table A.3 we present the results from the AES cryptographic functions evaluation.

HMAC Algorithm	**Data Size (bytes)**	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
SHA256-80	196	0.007095	0.003060	1.974	0.006	0.004241
	256	0.007130	0.001429	0.249	0.007	0.001980
	572	0.009113	0.001353	0.299	0.009	0.001875
SHA256-128	196	0.007100	0.001043	0.236	0.007	0.001445
	256	0.007106	0.001495	0.249	0.007	0.002073
	572	0.009154	0.001711	0.300	0.009	0.002371
SHA256-256	196	0.007085	0.000837	0.225	0.007	0.001160
	256	0.007106	0.001181	0.301	0.007	0.001636
	572	0.009176	0.002007	0.309	0.009	0.002781
BLAKE2b-80	196	0.012085	0.001621	0.689	0.012	0.002246
	256	0.012157	0.002596	0.783	0.012	0.003597
	572	0.017095	0.001666	0.286	0.016	0.002309
BLAKE2s-80	196	0.008108	0.001284	0.293	0.008	0.001779
	256	0.008114	0.001279	0.254	0.008	0.001773
	572	0.010114	0.005081	3.393	0.010	0.007042

Table A.1: HMAC Functions performances, using all size data inputs

Then we present the results from the evaluation of protocol specific functions, in this case R-

GMAC Algorithm	Data Size (bytes)	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
AES128-64	196	0.005023	0.000923	0.244	0.004	0.001278
	256	0.005079	0.000755	0.167	0.005	0.001047
	572	0.006140	0.001559	0.254	0.006	0.002160
AES128-128	196	0.004100	0.001040	0.232	0.004	0.001442
	256	0.004146	0.006897	3.438	0.004	0.009558
	572	0.006129	0.001757	0.578	0.006	0.002435
AES256-64	196	0.005103	0.001208	0.219	0.005	0.001674
	256	0.005079	0.000730	0.218	0.005	0.000975
	572	0.006181	0.001220	0.256	0.006	0.001689
AES256-128	196	0.004104	0.001111	0.238	0.004	0.001540
	256	0.004106	0.002030	1.103	0.004	0.002814
	572	0.006112	0.001716	0.311	0.006	0.002378

Table A.2: GMAC Functions performances, using all size data inputs

Enc/Dec Algorithm	Data Size (bytes)	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
AES256-Enc	51	0.005363	0.003738	2.141	0.005	0.005180
	204	0.005300	0.001788	0.205	0.005	0.002478
	408	0.015028	0.004485	0.338	0.013	0.006215
AES128-Enc	51	0.005403	0.002560	0.302	0.004	0.003548
	204	0.009107	0.003504	3.337	0.008	0.004856
	408	0.012948	0.004250	0.279	0.011	0.005889
AES256-Dec	51	0.005325	0.002072	0.300	0.004	0.002872
	204	0.010038	0.003260	0.260	0.009	0.004518
	408	0.014901	0.004338	0.289	0.013	0.006012
AES128-Dec	51	0.004340	0.002102	0.245	0.004	0.002913
	204	0.008111	0.003380	0.271	0.007	0.004684
	408	0.012765	0.004057	0.341	0.011	0.005622

Table A.3: AES Functions performances, using all size data inputs

GOOSE. Table A.4 presents the evaluation of InsertHMAC functions, Table A.5 presents the evaluation of ValidateHMAC, Table A.6 presents the evaluation of InsertGMAC and Table A.7 presents the evaluation of ValidateGMAC function. Finally, Table A.8 presents the performance evaluation of both R-GOOSE Payload Encryption and Decryption functions.

HMAC Algorithm	Data Size (bytes)	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
SHA256-80	1	0.009438	0.010469	1.260	0.009	0.014510
	20	0.010999	0.010252	1.018	0.010	0.014275
	220	0.018339	0.010861	0.864	0.017	0.015052
SHA256-128	1	0.009454	0.010347	1.157	0.009	0.014339
	20	0.010287	0.001429	1.057	0.010	0.014665
	220	0.017368	0.001353	0.927	0.017	0.014760
SHA256-256	1	0.009420	0.009784	1.070	0.009	0.013559
	20	0.010763	0.009487	0.932	0.010	0.013148
	220	0.017879	0.010569	0.888	0.017	0.014648
BLAKE2b-80	1	0.016356	0.011396	6.310	0.016	0.019347
	20	0.018312	0.011894	1.009	0.017	0.016484
	220	0.021435	0.011314	0.850	0.020	0.015680
BLAKE2s-80	1	0.011262	0.010724	1.003	0.010	0.014862
	20	0.011422	0.010808	0.951	0.011	0.014977
	220	0.021403	0.011867	0.847	0.020	0.016446

Table A.4: InsertHMAC Functions performances, using all size data inputs

HMAC Algorithm	Data Size (bytes)	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
SHA256-80	1	0.007144	0.007236	0.997	0.007	0.010028
	20	0.008205	0.008036	0.712	0.008	0.011137
	220	0.015231	0.008107	0.582	0.015	0.011235
SHA256-128	1	0.007162	0.007920	0.753	0.007	0.010980
	20	0.008154	0.006937	0.915	0.008	0.009610
	220	0.015226	0.007758	0.673	0.015	0.010750
SHA256-256	1	0.007680	0.007337	0.730	0.007	0.010168
	20	0.008169	0.007664	1.145	0.008	0.010622
	220	0.015235	0.008377	0.837	0.015	0.011610
BLAKE2b-80	1	0.013214	0.007349	0.595	0.013	0.010180
	20	0.014250	0.008113	0.590	0.014	0.011240
	220	0.026332	0.008524	2.368	0.026	0.011810
BLAKE2s-80	1	0.008161	0.007125	0.725	0.008	0.009870
	20	0.009185	0.008096	0.689	0.009	0.011220
	220	0.018285	0.008311	0.640	0.018	0.011520

Table A.5: ValidateHMAC Functions performances, using all size data inputs

GMAC Algorithm	Data Size (bytes)	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
AES128-64	1	0.007167	0.008984	1.344	0.007	0.011065
	20	0.007282	0.008822	1.027	0.007	0.012226
	220	0.014428	0.009849	1.000	0.014	0.013649
AES128-128	1	0.006202	0.007197	0.875	0.006	0.009975
	20	0.007239	0.010151	3.945	0.007	0.014068
	220	0.014357	0.009642	0.904	0.013	0.013363
AES256-64	1	0.007230	0.008474	1.023	0.007	0.011744
	20	0.008234	0.010012	2.821	0.007	0.013875
	220	0.014353	0.009982	1.276	0.014	0.013834
AES256-128	1	0.007187	0.014322	5.871	0.006	0.019849
	20	0.007454	0.008783	1.075	0.007	0.012172
	220	0.014504	0.011468	1.316	0.014	0.015894

Table A.6: InsertGMAC Functions performances, using all size data inputs

GMAC Algorithm	Data Size (bytes)	Average (ms)	Standard Deviation	Maximum (ms)	Minimum (ms)	Confidence Interval 95%
AES128-64	1	0.005128	0.006455	1.065	0.005	0.008946
	20	0.005595	0.006871	0.774	0.005	0.009523
	220	0.012295	0.008876	1.615	0.012	0.012301
AES128-128	1	0.005147	0.006555	0.830	0.005	0.009085
	20	0.005128	0.006355	0.852	0.005	0.008807
	220	0.012230	0.007611	0.808	0.011	0.010549
AES256-64	1	0.005111	0.005541	0.729	0.005	0.007679
	20	0.006109	0.006147	0.679	0.006	0.008520
	220	0.012196	0.007210	0.695	0.012	0.009992
AES256-128	1	0.005117	0.005593	0.703	0.005	0.007752
	20	0.005138	0.006141	0.704	0.005	0.008511
	220	0.012266	0.008412	1.590	0.011	0.011658

Table A.7: ValidateGMAC Functions performances, using all size data inputs

Enc/Dec Algorithm	**Data Size (bytes)**	**Average (ms)**	**Standard Deviation**	**Maximum (ms)**	**Minimum (ms)**	**Confidence Interval 95%**
AES256-Enc	1	0.011215	0.007693	0.639	0.010	0.010661
	20	0.013319	0.010709	3.941	0.013	0.014842
	220	0.034652	0.010920	1.009	0.034	0.015134
AES128-Enc	1	0.009238	0.007840	0.862	0.009	0.010865
	20	0.011233	0.008631	0.797	0.011	0.011962
	220	0.029677	0.011547	0.954	0.029	0.016003
AES256-Dec	1	0.008268	0.006511	0.745	0.008	0.009024
	20	0.011179	0.006844	0.677	0.011	0.009485
	220	0.032468	0.010302	3.285	0.032	0.014378
AES128-Dec	1	0.008719	0.006942	0.908	0.008	0.009621
	20	0.011192	0.006772	0.870	0.011	0.009386
	220	0.032388	0.007820	0.888	0.032	0.010837

Table A.8: Encryption/Decryption Functions performances, using all size data inputs