



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER DEGREE IN ICT FOR INTERNET AND MULTIMEDIA

Final Dissertation

**“An on-premises IoT platform for gas smart meter’s
management based on DLMS protocol”**

Supervisor

Prof. Lorenzo Vangelista

Master Candidate

Shahrzad Rafieitari

Academic Year: 2021-2022

Graduation Date: 11/07/2022

Abstract

Nowadays, gas, electricity, and water consumption in residential units can be tracked by smart meters using Internet of Things (IoT) technology. So, there is a need for a meter reading system that records information from the meters, which can be valuable for a variety of purposes.

The thesis considers the challenge of connecting smart meters (for gas, water, and electricity, but with particular emphasis on gas) to a central system in a private data center, where the data are used for various purposes, including diagnostics, billings, etc. Smart meters need a globally accepted standard that ensures security, interoperability, and efficiency. A widely used standard for this purpose is DLMS/COSEM (Device Language Message Specification)/ (Companion Specification for Energy Metering), which is used and explained in detail throughout the thesis.

An on-premises IoT platform for smart gas metering management has been developed based on DLMS/COSEM protocol. It demonstrates data exchange between a smart gas meter and a data acquisition system to manage and control the end customer's gas consumption. Smart meters are installed at the end customer's premises and send data to the IoT platform to which the gas suppliers and utilities have access. The developed IoT platform reports the data received from the meters, such as gas consumption, real-time alarms, and events elaborated by the gas meter. These collected data are COSEM objects that are stored in the local database. Moreover, the utilities can control (open/close) the valve of the gas meters using the IoT platform developed in this thesis.

The critical components of smart meters are security features. Thus, the latest security technologies are utilized in this project to protect smart gas meters.

Preface

This thesis is submitted as partial fulfillment of the requirements of the master's degree in Information and Communication Technology (ICT) Engineering at the Department of Information Engineering of the University of Padova. The duration of this thesis was six months in Shitek Technology Srl, an Italian hardware and software company specializing in measurement instrumentation, remote energy control, plant management, and heat metering.

The work carried out has led to the development of a user-friendly on-premises IoT platform for managing smart gas meters. This software is provided to Shitek Technology Srl and can be offered for sale by this company.

The project was supervised by Prof. Lorenzo Vangelista from the University of Padova. At Shitek Technology Srl, the project was supervised by Eng. Nicola Canella.

Acknowledgments

I would like to thank Prof. Lorenzo Vangelista for his great support and guidance during the project. Also, I would like to thank Eng. Nicola Canella and all the other members of the Shitek Technology Srl made my working period an enriching experience. Also, special thanks to the University of Padova for all its assistance during my studies.

Table of contents

Abstract	i
Preface	ii
Acknowledgments	ii
Table of contents	iii
List of tables	vi
List of figures	vii
Chapter 1: Introduction	1
1.1 Motivation and aims	1
1.2 Contribution	1
1.3 Background	4
1.3.1 DLMS Standardization	4
1.4 Thesis structure	5
Chapter 2: Framework	7
2.1 Smart meters	7
2.1.1 The benefits of smart meters	7
2.2 IoT platform for smart meters	8
2.2.1 Characteristics of an IoT platform for smart meters	8
2.3 Communication technologies for smart metering	8
2.3.1 Wired technologies for smart metering	8
2.3.2 Wireless technologies for smart metering	9
2.4 Data protocols used by smart meters	12
2.5 On-premises software vs cloud-based software	13
2.5.1 Which is better: On-premises or cloud?	13
2.5.2 On-premises software	14
2.5.3 Cloud-based software	14

2.5.4	Advantages of on-premises software	14
2.5.5	Disadvantages of on-premises software.....	14
2.5.6	Advantages of cloud-based software	15
2.5.7	Disadvantages of cloud-based software	15
Chapter 3: State of the art		16
3.1	DLMS/COSEM protocol.....	16
3.1.1	Physical and logical devices	16
3.1.2	Client/Server model	16
3.1.3	The Object Identification System (OBIS).....	17
3.1.4	The COSEM interface classes (ICs)	18
3.1.5	Accessing COSEM Interface Objects	20
3.1.6	DLMS/COSEM communication profiles.....	21
3.1.7	DLMS/COSEM security	23
Chapter 4: Project requirements		27
4.1	On-premises data center	27
4.2	Physical meter device	27
4.2.1	Meter device parameters	28
4.3	GSM/GPRS based communication network	28
4.4	Symmetric cryptography information	28
4.5	Development environment	28
4.6	Documentation	29
4.7	Gurux.DLMS library	29
4.8	Gurux GXDLMSDirector.....	29
4.9	Clear Terminal (CT)	29
Chapter 5: Project implementation.....		30
5.1	Connect the meter to the software by opening the port.....	30
5.2	Connect to the meter with “No Security”	30

5.3	Connect to the meter with High-Level Security + AES 128 encryption algorithm, and read the “Logical Device Name” of the meter and invocation counter	31
5.4	Read other remaining objects (converted volume, metering point id, ...)	32
5.5	Save the information in the database	33
5.6	Close the connection between the software and the meter	33
Chapter 6: Tests and deployments of the project		34
6.1	Log file translation	34
6.2	Database Diagrams	34
6.3	Developed IoT Platform displays	36
6.3.1	Login Page	37
6.3.2	Home Page	37
6.3.3	Smart Meters Page	40
6.3.4	Add Meters Page	41
6.3.5	Users Page	42
Chapter 7: Conclusion		44
Appendix A: Interface classes with their <i>class_id</i>		45
Appendix B: Codes related to the project implementation		47
B.1	Connect the meter to the software	47
B.2	DLMS connection with a “No Security”	48
B.3	High Level Security & encryption	51
B.4	Capturing the other required objects from the meter	53
B.5	Storing the data in database	60
B.6	Close connection and disconnect DLMS	61
Appendix C: DLMS messages translations		62
Appendix D: Abbreviations and acronyms		84
References		87

List of tables

Table 1: The structure of an interface class [5].....	19
Table 2: Authentication mechanisms of DLMS/COSEM [34]	24
Table 3: DLMS/COSEM security suite [37].....	25
Table 4: CurrentDiagnostic (Alarms) list [39].	32
Table 5: Usertbl-Users table of database.	43
Table B.2. 2: DLMS_IoT_Device table.....	51
Table B.2. 3: DLMS_Keys table.....	51
Table B.3. 1: COSEM logical device name object table [2].....	52
Table B.4. 1: Metering point ID object table [2].....	54
Table B.4. 2: Absolute converter volume object table [2].	54
Table B.4. 3: Current Diagnostic object table [2].	55
Table B.4. 4: Battery estimated remaining object table [2].	56
Table B.4. 5: Event log object table [2].	57
Table B.4. 6: Clock object table [39].	57
Table B.4. 7: Single Action Schedule object table [2].....	58
Table B.4. 8: Single Action Schedule object table [39].....	59
Table B.4. 9: Disconnect Control object table [2].	59

List of figures

Figure 1: Client/Server model [2].	17
Figure 2: OBIS code fields.	17
Figure 3: Communication session in the connection-oriented environment [10].	20
Figure 4: COSEM over IPv4 [10].	23
Figure 5: AT Commands included in the clear terminal.	29
Figure 6: Database datagrams.	34
Figure 7: User's datagrams in database.	35
Figure 8: Diagrams related to the IoT_Devices diagram in the database.	35
Figure 9: Three different smart gas meters were tested during the project.	36
Figure 10: The meter's display.	36
Figure 11: Set the meter.	36
Figure 12: Login page.	37
Figure 13: Smart Maters page of the IoT platform.	37
Figure 14: Alarms page of the IoT platform.	38
Figure 15: General section of the IoT platform.	38
Figure 16: Read Data section of the IoT platform.	39
Figure 17: Daily Volume section of the IoT platform.	39
Figure 18: Graphics section of the IoT platform.	40
Figure 19: Settings section of the IoT platform.	40
Figure 20: Information page of the IoT platform.	41
Figure 21: Edit Meter section of the IoT platform.	41
Figure 22: Add Meter page of the IoT platform.	42
Figure 23: Users page of the IoT platform.	43
Figure B.2. 1: "DLMS_IoT Devices" and "DLMS_Keys" diagrams.	50
Figure B.3. 1: DLMS_Connection_Sessions diagram in database.	53
Figure B.5.1: DLMS_Connection_Sessions and DLMS_Data_Records diagrams in the database.	60

Figure C. 1: AssociationRequest (AARQ).....	62
Figure C. 2: AssociationResponse (AARE).....	63
Figure C. 3: GetRequest for LDN.....	63
Figure C. 4: GetResponse for LDN.....	64
Figure C. 5: GetRequest for "Metering Point ID".....	64
Figure C. 6: GetResponse for "Metering Point ID".....	65
Figure C. 7: GetRequest for "Invocation Counter".....	65
Figure C. 8: GetResponse for "Invocation Counter".....	66
Figure C. 9: ReleaseRequest for disconnect.....	66
Figure C. 10: ReleaseResponse for disconnect.....	66
Figure C. 11: AssociationRequest (AARQ) for HighGMAC.....	67
Figure C. 12: AssociationResponse (AARE) for HighGMAC.....	68
Figure C. 13: ActionRequest for "Association LN" and replying the "HLS authentication".....	69
Figure C. 14: ActionResponse for "Association LN" and replying the "HLS authentication".....	69
Figure C. 15: GetRequest for LDN with HighGMAC.....	70
Figure C. 16: GetResponse for LDN with HighGMAC.....	70
Figure C. 17: GetRequest for "Metering Point ID" with HighGMAC.....	71
Figure C. 18: GetResponse for "Metering Point ID" with HighGMAC.....	71
Figure C. 19: GetRequest for "AbsoluteConverterVolume" with HighGMAC.....	72
Figure C. 20: GetResponse for "AbsoluteConverterVolume" with HighGMAC.....	72
Figure C. 21: GetRequest for the third attribute of "AbsoluteConverterVolume" with HighGMAC.....	73
Figure C. 22: GetResponse for the third attribute of "AbsoluteConverterVolume" with HighGMAC.....	73
Figure C. 23: GetRequest for "CurrentDiagnostic" with HighGMAC.....	74
Figure C. 24: GetResponse for " CurrentDiagnostic " with HighGMAC.....	74
Figure C. 25: GetRequest for "BatteryEstimatedRemainingUse_0" with HighGMAC.....	75
Figure C. 26: GetResponse for "BatteryEstimatedRemainingUse_0" with HighGMAC.....	75
Figure C. 27: GetRequest for the third attribute of "BatteryEstimatedRemainingUse_0" with HighGMAC.....	76
Figure C. 28: GetResponse for the third attribute of "BatteryEstimatedRemainingUse_0" with HighGMAC.....	76

Figure C. 29: GetRequest for " MetrologicalEventLogbook " with HighGMAC.	77
Figure C. 30: GetResponse for " MetrologicalEventLogbook " with HighGMAC.	77
Figure C. 31: GetRequest for the third attribute of " MetrologicalEventLogbook " with HighGMAC.	78
Figure C. 32: GetResponse for the third attribute of " MetrologicalEventLogbook " with HighGMAC.	78
Figure C. 33: GetRequest for the third attribute of " NonMetrologicalEventLogbook " with HighGMAC.	79
Figure C. 34: GetResponse for the third attribute of " NonMetrologicalEventLogbook " with HighGMAC.	79
Figure C. 35: GetRequest for the second attribute of " Clock" with HighGMAC.	80
Figure C. 36: GetResponse for the second attribute of "Clock" with HighGMAC.	80
Figure C. 37: SetRequest for the second attribute of " Single Action Schedule" with HighGMAC.	81
Figure C. 38: SetResponse for the second attribute of "Single Action Schedule" with HighGMAC.	81
Figure C. 39: SetRequest for the fourth attribute of "Single Action Schedule" with HighGMAC.	82
Figure C. 40: SetResponse for the fourth attribute of "Single Action Schedule" with HighGMAC.	82
Figure C. 41: GetRequest for the second attribute of "Disconnect Control" with HighGMAC.	83
Figure C. 42: GetResponse for the second attribute of "Disconnect Control" with HighGMAC.	83

Chapter 1: Introduction

1.1 Motivation and aims

Integrating smart meters in a city according to the Internet of Things (IoT) paradigm enables the recording of all necessary data for a smart city. Smart meters keep the city updated and ensure that each subsystem is functioning as intended. The goal is to achieve better gas, water, and electricity management that balances demand and consumption. As the population grows, so does the need for infrastructure for electricity, gas, water, etc. Due to this growing trend, utilities are forced to improve the quality of meter readings and, more importantly, the meter reading process. Smart meters can communicate bidirectionally once every three days with the utility's central system, which is an essential component of the advanced metering infrastructure (AMI). Thus, the utility can capture interval data, service interruptions, service restoration, time-based demand data, peak demand levels, and customer billing. In recent years, Advances in information and communication technology (ICT) have enabled the evolution from the most simple automatic meter reading system to AMI [1].

Utilities perform regular meter reading and billing as part of their regular operations.

The target of the thesis is to develop an on-premises IoT platform for remote management of gas meters installed in consumers' fields. This IoT device management platform must be able to:

- Acquire data (e.g., gas consumption), alarms (e.g., failed clock synchronization, software errors, flow errors, valve errors, tampering, memory errors), valve status, and battery charge status.
- Edit the configuration of the gas meter and send commands (e.g., closing the valve) to the meter.

By analyzing data and events collected from smart gas meters, utilities can identify consumers with unpaid bills or those who have recently relocated. In this case, the gas valve can be closed via the IoT platform. Additionally, utilities can utilize this platform in order to encourage consumers to reduce their consumption when gas quality, reliability, or prices are at risk. These companies can also offer consumers more targeted tariffs to reduce high-cost consumption.

1.2 Contribution

The IoT platform is an IoT solution for smart meter manufacturers. In this thesis, the IoT platform is developed, installed, and tested on a local server, and it can be accessed through a web browser. The IoT platform developed in this thesis is a web-based software or web application. The TCP/IP (Transmission Control Protocol/ Internet Protocol) is used to establish the connection between the smart gas meter and the software over IP-enabled networks such as GPRS (General Packet Radio Service), which is used by a GSM (Global System for Mobile communication) system as a low-cost 2G solution. Thus, data communication takes place via a cellular network that operates with the GSM system. The DLMS/COSEM (Device Language Message Specification)/ (Companion Specification for Energy Metering) communication protocol is used

for data exchange between the smart gas meter and the data acquisition system, which is the developed web-based software. The HTTPS (Hypertext Transfer Protocol Secure) is used to ensure secure communication over the network. Furthermore, the wrapper address is used in this work since the communication profile is TCP-UDP/IP, as described in Section 3.1.6.2.

The IoT platform in this work has flexible user access management, and providers use the same platform for multiple user roles. There is a workspace where device managers can log in with their dashboards and device management access rights. Only the dashboards, charts, and alerts are visible on the platform for customers.

To exchange data between the data acquisition system and the metering device, COSEM uses the client/server model. The client (web-based software) and the server (gas meter) establish a logical connection known as an application association (AA) to be aware of the exchange rules, such as:

- The application context
- The authentication context
- The xDLMS context

The application context specifies the short name (SN) or logical name (LN), which constitutes the referring style of COSEM attributes and methods. Additionally, it determines if ciphering is employed or not. Therefore, the application contexts [2], [3] are as follows:

- Logical Name without ciphering
- Short Name without ciphering
- Logical Name with ciphering
- Short Name with ciphering

An entity's authentication mechanism is determined by the authentication context. To identify the client or both the client and the server, an authentication mechanism can be used during the AA establishment. The available authentication levels are Lowest-Level-Security (No Security), in which neither the client nor the server is authenticated, Low-Level-Security, which authenticates only the client, and several variants of High-Level-Security, which authenticates both the client and the server [2]–[4]. In this thesis, Lowest-Level-Security is used to recognize the gas meter, and then “High-Level-Security” is used to read the gas meter information. Additionally, the “AES-128” algorithm and the “GMAC” ciphering are used to encrypt the DLMS packets during data exchange. Therefore, the authentication mechanism is “HLS GMAC” or “HighGMAC”. Moreover, the “AuthenticationEncryption” security level is used. The above authentication levels and security features are described in detail in Sections 3.1.7 and 5.3 and shown in Appendix B.3.

xDLMS contexts are responsible for determining the COSEM object-related services and capabilities to be used [3].

The target of the first stage of the software development is to successfully read the logical device name (LDN) of a gas meter by the public client. The purpose of the public client is mainly to retrieve the LDN of the gas meter. This resource can be accessed by the public client within an explicit application association (AA) without encryption. In other words, the software sends an association request (AARQ) to the meter and waits for the response (AARE), then the software sends a (COSEM GetRequest) to the meter to retrieve LDN and waits for an answer (COSEM GetResponse), and finally, the software sends a release request (RLRQ) to the meter to disconnect and waits for an answer (RLRE) (Refer to Appendix C).

Subsequently, data exchange must be performed by the management client to retrieve all required COSEM objects or data from the gas meter.

Once the software has identified the gas meter through its LDN, authentication can be performed using the management client. Consequently, the COSEM objects can be accessed within a pre-established association (without sending AARQ and RLRQ APDUs), and the messages are protected with an encryption key [2], [4], [5] (Refer to Section 3.1.7, Section 5.3, and Appendix C).

The software has the following features as shown in sections 6.2 and 6.3:

- A page that provides a list of all gas meters installed in the fields of the consumers with the following information for each gas meter:
 - Logical Device Name or Serial Number, which is mandatory to identify the gas meter during the data exchange.
 - Description, which contains the type of the meter and its manufacturer.
 - The address of the gas meter installation.
 - Absolute Corrected Volume, which is the entered gas volume in the meter.
 - Last reading of the “Absolute Corrected Volume” data.
 - The alarm status of the gas meters.
- A page for each installed gas meter, which contains the following sections:
 - The general information regarding each gas meter, such as the battery status, valve management, etc.
 - A list of the data or COSEM objects obtained from each gas meter.
 - A list of the daily corrected volume along with the chart.
 - The settings section, which is based on the meter and user information.
- A page containing detailed information captured from each meter as items below:
 - LDN (Logical Device Name) or Serial Number of each meter.
 - Model, manufacturer, and type of each meter.
 - Date and time of the last connection of the meter to the software.
 - The number of packets RX /TX and bytes TX /RX.
 - Edit and delete meter actions.

- A page that allows adding a new gas meter and the related information (LDN, encryption keys).
- A page that allows adding a new user and the related information.
- A database in which the encryption keys associated with the gas meter are stored. Without the encryption key, the DLMS/COSEM client cannot access most of the gas meter data. A database that stores the data received from each gas meter.

The software serves two kinds of agents:

- User: Using the CRUD (Create, Read, Update, Delete) pages to manage IoT devices, and read data and statistics.
- Gas meter (IoT device): Connects to the software, receives DLMS commands, and transmits DLMS objects.

Finally, the developed on-premises IoT platform would be able to communicate with multiple gas meters. This capability of the platform can increase interpretability and save time in the implementation of smart metering systems.

1.3 Background

1.3.1 DLMS Standardization

The Device Language Message Specification User Association (DLMS UA) is an international organization in the development and certification of interoperability in strategic energy management [6]. “DLMS UA” develops and maintains the DLMS specification, which has been adopted as part of the “IEC 62056 DLMS/COSEM” suite by “IEC TC13 WG14” [7], [8].

“IEC 62056” [7] is a collection of standards for the exchange of data for meter readings, electricity metering, tariffing, and load management established by the IEC (International Electrotechnical Commission) [5].

Following the adoption of the DLMS specification into the “IEC 62056”, the “IEC TC 13” [8] has also incorporated the DLMS color books into the “IEC 62056 series”. The “IEC 62056 series” contains the following standards, in which the contents of the "DLMS Green Book" and the "DLMS Blue Book" are reflected [5], [9]:

- IEC 62056-21 defines the direct exchange of local data.
- IEC 62056-42 defines the services and methods for asynchronous connection-oriented data exchange at the physical layer.
- IEC 62056-46 defines the data link layer based on HDLC protocol.
- IEC 62056-47 defines the COSEM transport layers of IPv4 networks.
- IEC 62056-53 defines the COSEM application layer.
- IEC 62056-61 defines the Object Identification System (OBIS).
- IEC 62056-62 defines the Interface Classes (IC).

DLMS (Device Language Message Specification) is the application layer protocol that turns the information contained in COSEM objects into messages. It provides services to connect

clients and servers and to access the data held by the COSEM objects. It also creates the messages (APDUs, Application Protocol Data Units), implements cryptographic protection if needed, checks and removes it, and manages the transmission of long messages in blocks [2].

Gas meters are modeled by COSEM (Companion Specification for Energy Metering) as a server application that is used by client applications that acquire data from the gas meter and invoke known actions within the gas meter using standard access methods to the COSEM objects. These objects are uniquely identified through a set of OBIS (Object Identification System) codes [2].

The specifications of DLMS/COSEM are organized in the form of colored books. The OBIS codes, interface classes, and methods to be implemented in the meter are described in the Blue Book. The communication profiles for communication through different channels are defined in the Green Book. In addition, the Green Book defines the procedures for data transmission between the client and the meter [2], [10]. Additionally, the Yellow Book provides a conformance test plan for COSEM objects. Moreover, a glossary of terms is included in the White Book [9].

Communication profiles define how the DLMS/COSEM application layer is supported by lower protocol layers [10] described in Section 3.1.6.

A security architecture for exchanging data between a client and a server is defined in DLMS [10]. In the thesis, various security features in DLMS/COSEM are also discussed in detail, including how these features are optimized for the authentication of the client, the server, and the data exchange.

1.4 Thesis structure

This thesis contains seven chapters. Chapter 1 presents the introduction of the thesis.

Chapter 2 describes the framework of the thesis, which contains the description of the smart meters and their advantages, the IoT platform for smart meters and its characteristics, and all types of communication technologies for smart meters with their instances. This chapter also compares on-premises and cloud-based software and identifies their advantages and disadvantages.

Chapter 3 presents the state of the art, including a description of the DLMS/COSEM protocol and all its associated details such as models, services, interface classes, communication profiles, and security considerations.

Chapter 4 describes all the requirements of the project that must be met.

Chapter 5 describes in detail the project steps regarding the connection of the meter to the software and reading the meter.

Chapter 6 provides a detailed description of the visible results, deployments, and tests of the project, such as a description of the translations of the DLMS messages, diagrams of the database, and pages of the developed IoT platform.

Eventually, Chapter 7 presents the conclusion of the thesis.

Additionally, this work includes three appendices.

Appendix A illustrates all the COSEM interface classes with their instances.

In continuation of Chapter 5 , Appendix B outlines and describes in detail the steps required to establish the connection between the meter and the software. In addition, it represents the steps necessary to read the meter information by obtaining all of the COSEM objects. Furthermore, this Appendix contains the database tables, interface classes, and OBIS codes used to exchange data between the meter and the software.

Appendix C completes Section 6.1 (Log File Translation) of Chapter 6 by demonstrating all translations of DLMS messages exchanged between the meter and the software, based on the steps described in Chapter 5 and Appendix B. These translations include all messages exchanged between the client and the server in order to establish the connection between the server (meter) and the client (software), as well as to read the meter information by reading and setting the attributes of the COSEM objects.

Finally, Appendix D contains the table of abbreviations and acronyms used in this thesis.

Chapter 2: Framework

2.1 Smart meters

In recent years, smart meters have spread all over the world. The smart meter is an intelligent digital device designed as an optimal alternative to the traditional meter [11], [12]. The smart meter is an IoT device that records and monitors a consumer's energy consumption. Normally, smart meters measure energy consumption in near real-time and report it regularly at short intervals. As utility companies seek more accurate and timely statistics to improve their business, the growth of smart meters is increasing every day. Implementing two-way data communication between meters and the central system requires advanced metering infrastructure (AMI), a system for measuring, collecting, and analyzing energy consumption that simultaneously communicates with metering devices such as gas meters on a schedule [1], [13]. The bidirectional meter communication provided by AMI enhances automatic meter reading (AMR), a technology for collecting consumption data from energy meters and transmitting this data to a central database for troubleshooting, billing, and other services [14].

2.1.1 The benefits of smart meters

Tracking energy with smart meters creates a better understanding of how energy is being consumed. This allows consumers to see if the changes they have made to improve energy consumption are effective. Utilities can encourage their customers to make smart energy consumption decisions based on accurate gas or water usage statistics generated by smart meter technology. The key benefits of smart meters are as follows:

- **Convenience**

In contrast to the traditional and manual process, smart meters automatically send readings of consumers' energy consumption to utilities for billing purposes [15].

- **Accuracy**

Consumers can receive accurate bills because smart meters transmit numbers directly to utilities at fixed dates and times. So, there are no human errors in billing [15].

- **Control over energy consumption**

Consumers can use energy-saving appliances to save money by comparing monthly costs based on statistics reported to utilities by smart meters [15].

- **Safety from faulty appliances**

Consumers can detect faulty equipment through the built-in displays equipped with smart meters because by displaying the energy consumption at a certain time, a sudden increase in equipment can be detected, which not only maintains safety but also leads to optimal energy consumption [15].

- **Environment protection**

As the use of smart meters reduces customers' energy needs, it can also reduce the need to build power plants [15].

2.2 IoT platform for smart meters

One of the biggest challenges in smart meter implementation is integrating smart meters into their infrastructures and organizing customized smart metering applications. Integrating smart meters into their infrastructure requires IoT platforms that can process data as a meaningful smart metering solution [16].

A suitable IoT platform can monitor smart meters in real-time to track energy consumption and send metering data to utilities for scheduling and accurate billing. IoT platforms as IoT solutions can conserve a lot of energy and provide enormous value to customers and the environment [16].

2.2.1 Characteristics of an IoT platform for smart meters

- **Security**

The entire security vulnerability should be detected by the IoT platform, and its influence must be prevented to keep the information safe. Both the communication channels and the data exchanged between smart meters and users should be secure [17], [18].

- **Scalability**

Scaling the IoT platform should grow with the evolution of the business while maintaining the integrity, security, reliability, and performance of the solution [17], [18].

- **Reliability**

The reliability of the platform in terms of service delivery is critical, such as the timely provision of solutions to address outages [17], [18].

- **Interpretability**

The platform must be able to run, schedule, and communicate with any meter, and it must be designed to connect without hardware [17], [18].

- **Modularity**

As the rapid deployment of an IoT platform is achieved through modularity in different ways to meet business-specific requirements, modularity is critical to the implementation of an IoT solution [17], [18].

2.3 Communication technologies for smart metering

Communication technologies for smart meters are divided into wired and wireless technologies [19], [20].

2.3.1 Wired technologies for smart metering

There are three non-wireless technologies used in smart meters:

- **Power Line Communication (PLC)**

PLC (Power Line Communication) uses existing power lines installations in communication. This offers the advantage of using the current widespread infrastructure without the need to lay special cables. The installation of PLC modules in meters is quite simple, making this technology the most popular.

The main disadvantage of PLC is the attenuation due to random switching of electrical devices, which can result in changes in power parameters in the power distribution network [19], [20].

- **Digital Subscriber Line (DSL)**

Data can be transmitted over conventional telephone lines with Digital Subscriber Line (DSL) as a communication technology. It is quite reliable and inexpensive, as the infrastructure is already in place in most cases. However, as the distance between the supplier and the consumer increases, the throughput decreases [19].

- **Fiber Optic Communications**

Due to high implementation costs, this is not a very popular option for smart metering, but it is still worth mentioning for use cases where high data rates are required. In various regions of the United States, Europe, and other parts of the world, fully fiber-based networks are still being built that often driving up costs.

In this work, fiber optic communication is used instead of the wrapper (refer to Section 3.1.6.2) in the case of using the HDLC (High-Level Data Link Control) communication profile [19].

2.3.2 Wireless technologies for smart metering

- **Radio Frequency Mesh (RF Mesh)**

This technology provides the core function of automatic meter reading (AMR) by enabling wireless communication. It is used to measure energy consumption. It provides better accuracy and coverage when combined with PLC. The use of Radio Frequency Mesh results in low cost and energy-efficient operation due to low power connectivity. This technology is only suitable for limited areas where there is a high concentration of RF modules [19].

- **Zigbee**

Zigbee is an interesting solution for low-power smart meters due to its low implementation costs and simplicity. While Zigbee resembles RF Mesh, it operates slightly differently. Therefore, it can be used to create a mesh network that connects smart meters to data concentrators rather than connecting smart meters to control centers. The high interference of applications using the same bandwidth is known as a disadvantage of Zigbee [19], [21].

- **WiMax**

WiMAX (Worldwide Interoperability for Microwave Access) is a standard for wireless communications. This technology is less well known in the field of metering and is simply ignored by many utilities due to insufficient bandwidth when serving a large number of clients [19], [21].

2.3.2.1 Cellular technologies for smart metering

Cellular technologies, as a subset of wireless technologies, are becoming increasingly important in the deployment of smart metering since they can enhance existing smart grid efforts [19]. By leveraging cellular IoT, there is no need to build infrastructure, as utilities can rely on the networks of mobile operators. With the outsourcing of the communications network, the installation costs and times of smart metering systems are significantly reduced. Moreover, cellular networks can support machine-to-machine (M2M) communication directly or via gateways. As a result, the M2M subscriber market is improved, leading to higher adoption rates, lower costs, and increased competition among companies [19], [21], [22].

- **GSM/GPRS**

Most of the cellular point-to-point infrastructures of smart metering systems are currently integrated with the Global System for Mobile Communications (GSM), which uses the General Packet Radio Service (GPRS) data service. A GPRS network is designed to transfer data packets specifically for Internet connections. Therefore, it can support TCP/IP communication.

The following protocols are supported by the GPRS:

- Internet protocol (IP)
- Point-to-point protocol (PPP)
- X.25 connection-removed

The X.25 protocol used by wireless payment terminals is no longer supported by GPRS. However, it can be supported via PPP or IP.

The IP protocol is the most significant component of GPRS. Every operator can support this protocol, and it provides internet connectivity to all devices through TCP. In GPRS networks, IP mobility is essentially handled like Mobile IP [19], [21].

A GSM modem allows data collection systems to communicate with GSM networks via a SIM (Subscriber Identity Module). 2G cellular networks typically offer data services with net data rates of up to a few tens of kbps. These data services are optimally suited for remote metering applications. In this work, GSM cellular technology is used for TCP/IP connection [19], [21].

- **Narrowband IoT (NB-IoT) for Smart Metering**

NB-IoT provides widespread cellular connectivity as part of the LPWAN (Low-power wide-area network) technology group, and it is a direct competitor of LoRaWAN [23]. The cost of this technology is lower than other M2M (Machine to Machine) technologies on the market. Its global network coverage is constantly evolves [19], [21].

As a result of integrating NB-IoT with a PLC/RF mesh infrastructure, meters can be acquired. Hence, this proves to be highly beneficial for smart metering as it eliminates the need for manual meter readings in remote areas, eliminates the need for concentrators and gateway devices, and allows for easy visualization of data [19], [21].

- **LTE for Machine Type Communication (LTE-M)**

Due to higher energy consumption, this technology is less popular than NB -IoT in terms of smart metering. Both data latency and data throughput are significantly improved with this technology [19], [21].

2.3.2.1.1 Requirements of cellular technologies for smart metering

The deployment of smart meters is an expensive, long-term project that requires a large capital investment. To be successful, they must meet several requirements as follow:

- **High reliability and long lifespan**

Ideally, smart meters and their supporting communication should be able to remain online for several years. As a result, they must be highly reliable. A variety of technologies are now available to meet smart metering requirements. For example, in GSM mobile phones, the plastic SIM (Subscriber Identity Module) is replaced by a ruggedized version integrated into the communication module [21].

- **Low maintenance**

The firmware and communication module protocols of a smart meter are likely to be updated over a period of 10 to 20 years. By using secure wireless device management services, utilities can easily manage, configure, test, validate and update firmware and application software remotely [21].

- **Cost effectiveness**

As smart metering solutions become more widespread, it is imperative that they are designed to minimize capital and operating costs. Technologies are available to embed protocol and application software into the communications module. Through this integration, the number of processors required, and the amount of memory utilized can be reduced. Thus, the overall cost is reduced [21].

- **Security**

Security threats must be considered when two-way command and control systems are integrated into energy management systems. They include the privacy of consumers, data integrity, detection of network jamming attacks, and ensuring service continuity [21].

- **Low installation costs**

Installing smart meters accounts for 30 percent of the cost of the system. As a result, providers require solutions that facilitate installation and reduce the time and expertise required. The development of cellular communication modules can be simplified with integrated network analysis tools, enabling faster and more efficient deployment and saving time and money [21].

- **Low power consumption**

The communication module in a modular design is powered by the electricity meter or, in the case of water and gas meters, by a battery. This means that certain design requirements must be met. The communication module must first be designed to operate at the low current that the meter is able to supply. The second issue is to ensure that the communication systems are highly energy-efficient, as even when smart meters and their communication systems consume relatively little power, this consumption amounts to quite a bit when millions of meters are connected point-to-point [21].

2.4 Data protocols used by smart meters

The communication protocols used by smart meters may differ by region. These protocols include:

- **DLMS/COSEM**

DLMS/COSEM (Device Language Message Specification)/ (Companion Specification for Energy Metering) are both parts contained in IEC 62056 (International Electrotechnical Commission), a set of international standards for smart meters described in Section 1.3.1. COSEM represents smart meter data and specifies its attributes through object modeling. DLMS is the application layer protocol that converts the information contained in the objects into messages. The DLMS/COSEM protocol is used in this project [20], [24].

- **ANSI C12.18**

ANSI C12.18(American National Standards Institute) is used primarily in North American countries. This standard was developed by the metering device for two-way communication with an ANSI Type 2 Optical Port, which is a connector for communication [20].

- **OSGP**

OSGP (Open Smart Grid Protocol) is the dominant group of standards published by the European Telecommunications Standards (ETSI) for the efficient transmission of commands to smart meters. This protocol uses the OSI (Open Systems Interconnection) model to solve the problems that are developing in the smart grid [20].

- **TCP/IP**

TCP/IP (Transmission Control Protocol/Internet Protocol) is a connection-oriented transport layer protocol that enables orderly and continuous data transmission between applications on Internet hosts. TCP is the dominant transport layer protocol on the Internet, and many important applications benefit from its service. TCP overcame significant challenges as the Internet evolved beyond its original features. The Internet of Things (IoT) is known as a new challenge for TCP. This is a network trend that envisions tens of billions of low-cost devices attached to everyday objects being connected to the Internet to enable smart scenarios. However, IoT devices typically have significant limitations, use low-rate connections, and are error-prone, and their networks often follow a multihop topology or multihop routing. Due to these harsh network conditions,

TCP has often been heavily criticized as a transport layer protocol for IoT [25]. In this project, the meter is connected to the IoT platform using the TCP/IP connection.

- **UDP/IP**

UDP/IP (User Datagram Protocol / Internet Protocol), unlike TCP / IP, prioritizes speed over accuracy and is an alternative to TCP / IP. This protocol reduces latency but is not able to correct transmission errors, such as packets that are out of sequence, or missing and repeating packets. UDP / IP may be increasingly used in smart meters in the future as the industry gets closer to real-time transmission [20].

- **MQTT**

MQTT (Message Queuing Telemetry Transport) is a protocol that requires very little bandwidth or network resources to transfer data between network entities. For this reason, it is often used in IoT applications. It is a lightweight publishing protocol (pub/sub) that is often paired with TCP/IP. In this publish/subscribe model used by MQTT, messages are "published" by smart meters, and these messages are distributed by the MQTT server to all network entities that have "subscribed" to the message type [20].

- **HTTP**

HTTP (Hypertext Transfer Protocol) is the basis of data communication on the World Wide Web and thus the most used protocol for navigating the Internet. Since this protocol was developed for one-to-one communication, it is not suitable for a smart meter for gateway communication. This protocol is paired with TCP/IP for data transport [20]. In this project, the HTTPS (Hypertext Transfer Protocol Secure) is used on the internet.

- **WebSockets**

WebSocket is a communication protocol used for communication between a client and a server. Full-duplex communication channels or two-way communication is provided by Websockets over a single TCP connection. This protocol can stream messages in real-time, but with high power consumption for battery-powered devices [20].

- **XMPP**

XMPP (Extensible Messaging and Presence Protocol) is an open XML (Extensible Markup Language) technology that enhances a wide range of real-time applications such as instant messaging. This application is defined in the application layer. Structured data between two or more network entities can also be exchanged in near real-time based on XML using XMPP [20].

2.5 On-premises software vs cloud-based software

2.5.1 Which is better: On-premises or cloud?

This is the question most established companies ask themselves whether it's worth moving to the cloud or investing in on-premises systems. So, they should know the differences between on-premises and cloud-based services and infrastructure. They should also consider the needs of their business to choose between cloud and on-premises services. Storing data on external or internal

servers can be one of the most important decisions for companies. In this project, an on-premises IoT platform is developed [26].

2.5.2 On-premises software

On-premises software is installed and run on computers within an organization. This software consists of a combined database and modules tailored to the specific needs of companies. On-premises software is set up in the company's internal system with other infrastructure required for the software to function. The responsibility for the operation and maintenance of the system lies with the company using the on-premises software [27].

2.5.3 Cloud-based software

A cloud-based environment is a type of software that is not deployed on-premises and is often known as "Software as a Service" ("SaaS") [27]. SaaS (Software as a Service) allows users to access software applications running on shared computing resources over the Internet. The computing resources are managed and maintained in remote data centers responsible for hosting different applications on different platforms [28].

2.5.4 Advantages of on-premises software

- **Better Customization**

As on-premises software is managed by enterprises, it can be customized to greater than a cloud-based service. Data is stored on the company's local server, which allows the owner to have complete control over the data [26].

- **Greater security**

The primary reason why companies choose to stay with on-premises software is to ensure security. Due to the high level of security, it is much less likely that others will be able to access the information. One of the main advantages of an on-premises license is the fact that the data entered into the software by the user can remain on the device or local server and is not sent to a third party [26], [29].

- **License purchase**

The on-premises license software is installed and used on the company's local server. The owner of the company can purchase this software once to have it forever, and it can be maintained manually by the owner [29].

2.5.5 Disadvantages of on-premises software

- **Scalability**

On-premises software does not scale as well as cloud-based software. For example, if the number of users of a program is increased, the software or hardware must be manually installed by IT staff to allow new users to utilize it [26].

- **Remote offices**

An organization with a large mobile workforce or several branch offices should provide its employees remote access through the on-site software, which would require an additional network, resulting in higher operating costs [26].

- **Initial costs**

If a company wants to buy the software and integrate it into its local system, the initial cost of services is high. It is even possible that with the appearance of new and updated programs on the market and the insufficient use of a program, the entire initial cost will be lost [26].

2.5.6 Advantages of cloud-based software

- **Affordability**

Companies with cloud-based software, rather than paying a large upfront licensing fee, will have much lower monthly fees. Therefore, the total cost of cloud-based applications can be considered lower. A monthly fee is charged for applications, which is the same as subscription fees offered by some providers, which include maintenance and support [26].

- **Ease of deployment**

One of the most important advantages of cloud computing is its rapid development capability, which allows customers to use the program within minutes and without long installation processes [26].

- **Management services**

There is no need for customers to manage the software or hardware. Instead, they can engage a vendor to manage all of this externally. By doing so, they can free up their employees and reduce costs [26].

2.5.7 Disadvantages of cloud-based software

- **Technical issues**

The user's access to the cloud may be limited due to technical issues such as power outages, slow internet connections, or disconnection of the device the user uses to connect to the cloud. These problems may also prevent the recovery of files from the cloud [30].

- **Less flexibility**

Since software vendors do not offer their customers a wide range of customizable options in the software via the cloud, they are offered the same schema for all consumers. So, one of the main problems for companies using cloud-based software is flexibility and customizability [26].

- **Security concerns**

Security breaches can occur in the cloud, such as data loss and leakage, or account hijacking, because hackers can access the company's data stored on the Internet. For this reason, enterprise migration to cloud-based systems has become a concern for some companies [30].

Chapter 3: State of the art

3.1 DLMS/COSEM protocol

Concerning Section 1.3, to exchange data with metering equipment, the DLMS/COSEM (Device Language Message Specification)/(Companion Specification for Energy Metering) specification defines the communication protocols and an interface model by following three steps:

- **Modeling**

The modeling step focuses on the data model of the meter and the rules for identifying the data. A detailed explanation of the meter's functions is provided by the data model [2].

- **Messaging**

Communication services and protocols are defined in the messaging step in order to map the data model elements to application protocol data units (APDUs) [2].

- **Transporting**

In this step, the services and protocols for transporting messages over the communication channel are covered [2].

3.1.1 Physical and logical devices

Physical meters are modeled by COSEM as logical devices, and each device has a Logical Device Name (LDN), which is a globally unique identifier, an octet string of up to 16 octets starting with the three-octet manufacturer's identifier.

A logical device operates as an application process (AP) and contains a set of “interface objects”, each containing a set of attributes and methods. For each logical device, the interface objects model a subset of the meter's functions, which is visible from the client side via the meter’s communication interfaces. The interface objects are specifically designed for the field of consumption measurement [2], [5].

3.1.2 Client/Server model

To exchange data between metering devices and data collection systems, COSEM models the metering devices as servers and the data acquisition systems as clients by using the client/server model. Thus, the data collection system can be referred to as a client AP (Application Process) and a metering device as a server AP [5], [10].

“Service requests” are sent from the client AP to the server AP, and the server AP sends “service responses” to the client AP. In addition, the server AP is able to make “unsolicited service requests” to notify the client AP of events or send data on pre-configured conditions.

The Open Systems Interaction (OSI) model is also used by DLMS/COSEM for data exchange between metering devices and data collection systems [5], [10].

Client AP and server AP are generally located on separate devices. Therefore, messages are exchanged through a protocol stack, as shown in Figure 1:

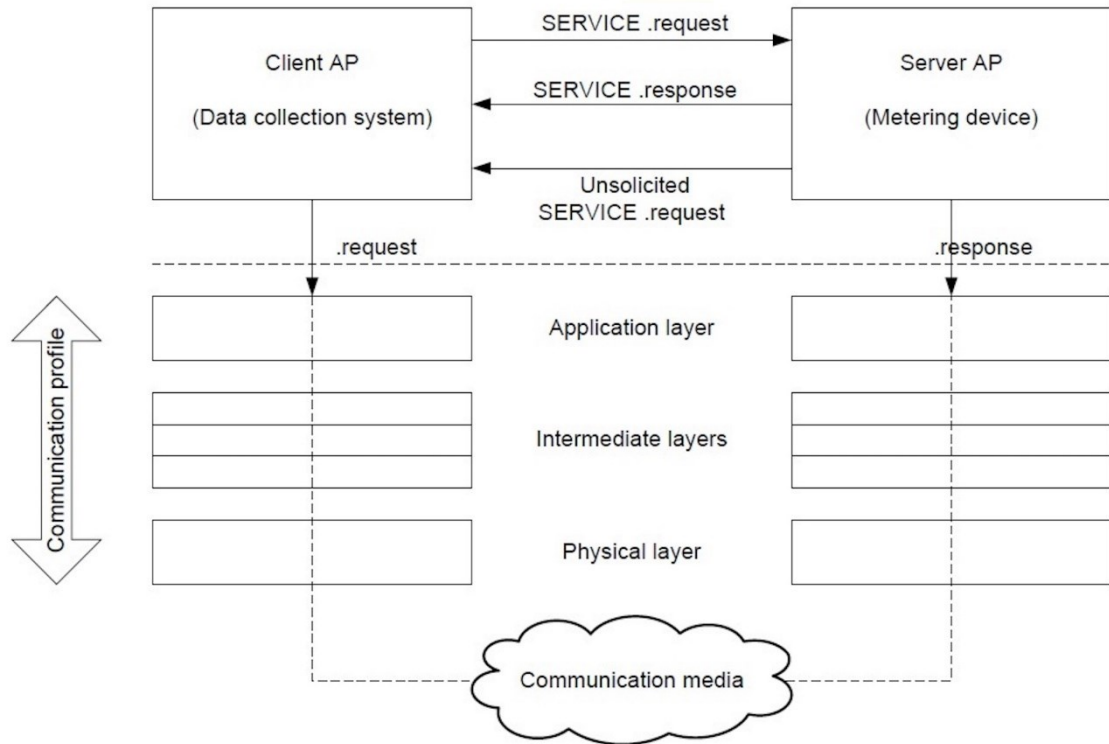


Figure 1: Client/Server model [2].

3.1.3 The Object Identification System (OBIS)

For identifying information in energy meters, each data element is assigned an identifier called “OBIS code”, which is divided into six value groups between A and F in a hierarchical structure [2]. The identification of the following items is done by OBIS codes [5]:

- Logical names of the different COSEM objects,
- Transmission of data over the communication lines,
- Data displayed on the device meters.

All OBIS codes are specified in the “DLMS Blue Book”[2] and the “IEC 62 056-61 Book”[31].

Figure 2 shows the fields of OBIS codes:

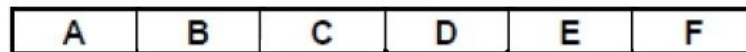


Figure 2: OBIS code fields.

The following ranges are available in the value groups B to F for manufacturer-specific purposes:

- Group B with the range from 128 to 199
- Group C with the range from 128 to 199, 240
- Group D with the range from 128 to 254
- Group E with the range from 128 to 254
- Group F with the range from 128 to 254

Group A: Value group A identifies the type of energy used in the metering. For example, gas, electricity, hot or cold water, heating, and cooling. Its range is 0-15 [2].

Group B: In value group B, the measurement channel number, the communications channel, and the multiple inputs in the meter are defined [2].

Group C: In value group C, the physical data elements are identified, such as voltage, temperature, current, volume, and power. These elements are determined by the value within the value group A [2].

Group D: Identified quantities in value groups A to C can be classified further by allocating them to value group D [2].

Group E: In this value group, various algorithms are utilized to determine the subsequent processes specified in the value groups A to D. These algorithms can provide the physical quantities, the energy, and the demand quantities [2].

Group F: The data values indicated by the value groups A to E are defined in this value group according to different billing periods [2].

3.1.4 The COSEM interface classes (ICs)

As mentioned in Section 3.1.1, each interface object consists of a set of attributes and methods. The properties of the object are represented by the *attributes*, and the behavior of an object is affected by the *value* of an *attribute*. Interface classes can be defined by their *name*, *class_id*, and *version methods* (Refer to Table 1). COSEM interface classes are entities with a set of methods and attributes that represent a specific function independently or in conjunction with other COSEM interface classes [10]. By defining each interface class (IC), each attribute is assigned an index [2], [5], [32].

3.1.4.1 Referencing methods

It is possible to refer to COSEM objects' attributes and methods in two different ways:

- **Using logical names (LN Referencing)**

The attributes and methods, in this case, are referenced by the unique identifier of the COSEM object they belong to. For each attribute, there are references such as *class_id*, the value of the *logical_name* attribute, and *attribute_index* (Refer to Table 1). The identifier of the *attribute* is the *attribute_index*, which is specified in the definition of each interface class.

For each method also there are references such as *class_id*, the value of *logical_name* attribute, and *method_index* (Refer to Table 1). Also, *Method_index* is used to identify the required method as the identifier, which is specified in the definition of each interface class [10].

- **Using short name (SN Referencing)**

By using this referencing, the attributes and methods can be identified by a short 13-bit integer. To simplify access to metering devices, some short_names have been reserved to be used

as base_names for special COSEM objects. In simple devices, this type of referencing is intended for use [10].

3.1.4.2 The structure of the interface class

Continuation of Section 3.1.4, Within a given interface class, the common attributes and methods are defined once for all objects. The proprietary attributes and methods of each object can be added by the manufacturers [2], [5].

Table 1 shows the structure of an interface class:

Class name		Cardinality	class_id, version			
Attribute(s)		Data type	Min.	Max.	Def.	Short name
1.	logical_name (static)	octet-string				x
2.	...	(...)				x + 0x...
3.	...	(...)				x + 0x...
Specific methods (if required)		m/o				
1.		...				x + 0x...
2.		...				x + 0x...
3.		...				x + 0x...

Table 1: The structure of an interface class [5].

Each interface class includes “class name”, “cardinality”, “class_id”, “versions”, “attributes”, and “methods” [2], [32].

- **Class name:** It identifies the interface class.
- **Cardinality:** The number of interface class instances in a logical device is specified with cardinality.
- **Class_id:** It is the interface class identification code, and its range is (0 to 65,535).
- **Version:** It means the identification code of the interface class version.

Each attribute contains a name, a data type, a minimum, a maximum, the default value of the attribute, and the “short name” [2], [32].

- **Attribute’s name:** it specifies the attributes belonging to an interface class.
- **Logical_name:** It must be the first mandatory attribute of each object. Its value can be an OBIS code or a short name.
- **Data type:** specifies the attribute’s data type.
- **Min.:** It indicates whether the attribute contains a minimum value. If it shows “X”, it means the attribute contains the minimum value, and if it shows <empty>, it means that the attribute does not have a minimum value.
- **Max.:** It indicates whether the attribute contains a maximum value. If it shows “X”, it means the attribute contains the maximum value, and if it shows <empty>, it means that the attribute does not have a maximum value.
- **Def.:** It indicates whether the attribute contains a default value.
- **Short name:** Using Short Name (SN) referencing, each attribute and method of each instance of an object is assigned a short name.

Each method contains a name indicating the specific methods that belong to the object, and m/o indicating whether the method is mandatory (m) or optional (o), method description.

The interface classes are divided into different groups such as “Data storage”, “Access control and management”, “Time and event bound control”, “Payment metering”, and so on. They are specified in the “DLMS Blue Book” and the “IEC 62056-62 Book” [2], [32]. All interface classes with their *class_id* can be found in Figures A.1, A.2, and A.3 of Appendix A.

3.1.5 Accessing COSEM Interface Objects

3.1.5.1 The concept of Application Association (AA)

The "Application Association" (AA) is defined by DLMS-UA to permit the client to access the "COSEM interface objects" within the server [9]. AA as an application-level connection is established between a client Application Process (AP) and a COSEM logical device as a server Application Process (AP) within the context of the COSEM client/server model [10] as described in Section 3.1.2.

AAs can be established in advance or at the client's request. During the establishment of AA, the exchange of the context data is initiated, and authentication mechanisms are chosen. After establishing AA, application data can be exchanged between the client AP and the server AP. Thus, the client AP has access to the COSEM interface objects on the servers. Then the AA is released after the data exchange [9], [10]. Therefore, the communication session in the connection-oriented environment has three phases, namely, “AA establishment”, “Message exchange”, and “AA release” as shown in the figure below:

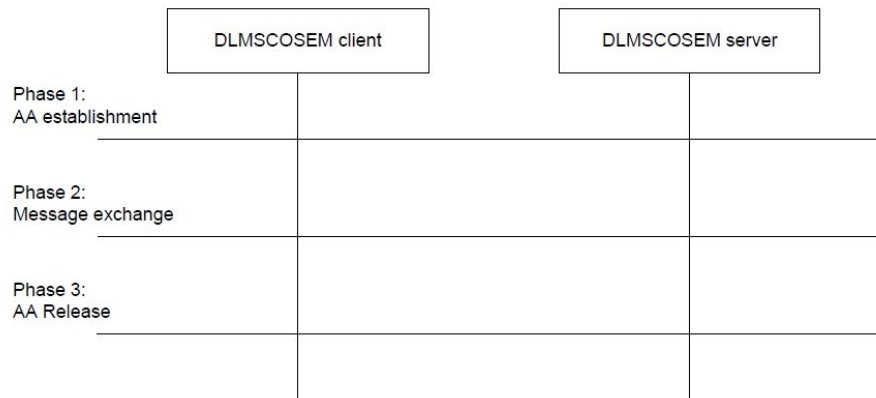


Figure 3: Communication session in the connection-oriented environment [10].

3.1.5.2 Data Communication Services

In DLMS/COSEM, client APs (Application Processes) always use logical names (LN Referencing) to refer to the data communication services. On the other hand, the server AP refers to the data communication services using either a logical name or a short name [5], [10].

Thus, in continuation of Section 3.1.2, the client/server services are defined by DLMS/COSEM for LN Referencing as follows:

- **Get service:** The attributes of COSEM interface objects are retrieved by this service [5], [10].
- **Set service:** The attributes of COSEM interface objects are modified by this service [5], [10].
- **Action service:** The attributes of COSEM interface objects are invoked by this service [5], [10].
- **EventNotification service:** Through this service, an unsolicited notification can be sent from the server to the client when an event occurs [5], [10].

In addition, the client/server services defined by the DLMS/COSEM application layer for SN Referencing are as follows:

- **Read:** Through the Read service, the client AP sends a request to the server AP to invoke the methods or retrieve the value of the attributes. [5], [10].
- **Write:** Through the Write service, the client AP sends a request to the server AP to invoke the methods or substitution of content in attributes [5], [10].
- **Unconfirmed write operations:** These operations are identical to the “Write” service, but unconfirmed [5], [10].

3.1.6 DLMS/COSEM communication profiles

A DLMS/COSEM communication profile consists of a specific set of protocol layers and the COSEM application layer (AL) [5].

The COSEM application layer consists of three mandatory components on both the client and server sides as follows:

- **The Association Control Service Element (ACSE).**

ACSE provides the services for establishing, maintaining, and releasing application associations (AAs). The services provided by ACSE [10], [33] are as follows:

- A-Associate Request or AARQ
- A-Associate Response or AARE
- A-Release Request or RLRQ
- A-Release Response or RLRE

- **The extended DLMS Application Service Element (xDLMS ASE).**

xDLMS ASE provides the data communication services between client and server application processes (APs) [10], [33] described in Section 3.1.5.

- **The Control Function (CF).**

CF specifies how the invocation of the corresponding service primitives of ACSE is controlled [10], [33].

Following Section 3.1.6, each DLMS/COSEM communication profile is distinguished by the protocol layers it contains, as well as the type of Application Control Service Element (ACSE)

within the application layer. To facilitate data exchange using various communication media, a single device may support multiple communication profiles [5].

Two communication profiles are specified by DLMS as follows:

3.1.6.1 HDLC-based communication profile

There are three layers in the connection-oriented HDLC-based profile, including a physical layer or serial connection, an HDLC layer, and an application layer. The 3-layer HDLC-based communication profile supports local data exchange with meters either directly via the optical port or a physical current loop interface or remote data exchange via the PSTN (Public Switched Telephone Network) [2], [5], [10].

In the HDLC-based communication profile, The Mac address of the client is a byte value, and the Mac address of the server is divided into two parts such as the logical device address, which is the upper part, and the physical device address, which is the lower part [2], [5], [10].

3.1.6.2 TCP-UDP/IP based communication profile

Data exchange can be supported by TCP-UDP/IP based communication profiles over the Internet through different physical media such as ISDN (Integrated Services Digital Network), GPRS (General Packet Radio Service), Ethernet¹, PSTN, or GSM (General Packet Radio Service) with PPP (Point-to-Point Protocol), etc. [5].

In TCP-UDP/IP based communication profiles, DLMS/COSEM AL (Application Layer) is supported by DLMS/COSEM TL (Transport Layer), which includes the internet TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) protocol as well as a wrapper. The wrapper ensures that the service set of DLMS/COSEM TL in the style of OSI is adapted to both TCP and UDP function invocations. Furthermore, it helps address logical devices by binding all of them to an SAP (Service Access Point) called a wrapper port (WPort). Lastly, the wrapper provides information regarding the length of the transmitted APDUs [5], [10]. TCP-UDP/IP based communication profiles and wrapper are used in this work.

The COSEM communication profiles based on TCP-UDP /IP contain five protocol layers such as [10]:

- DLMS/COSEM application layer.
- DLMS/COSEM transport layer.
- Network layer containing the Internet Protocol (IPv4 or IPv6).
- Data link layer (any data link protocol that supports the network layer).
- Physical layer (any physical layer that is provided by the data link layer).

¹ Ethernet is a standardized technology for local networks.

DLMS/COSEM AL (Application Layer) utilizes TCP or UDP services through a wrapper protocol in which the IPv4 or IPv6 network layers are used for communication between the nodes connected to this abstract network. In this environment, DLMS/COSEM AL is considered as another standard Internet application protocol that can coexist with other Internet application protocols such as HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), etc. [10]. Figure 5 shows the COSEM over the IPv4 network layer:

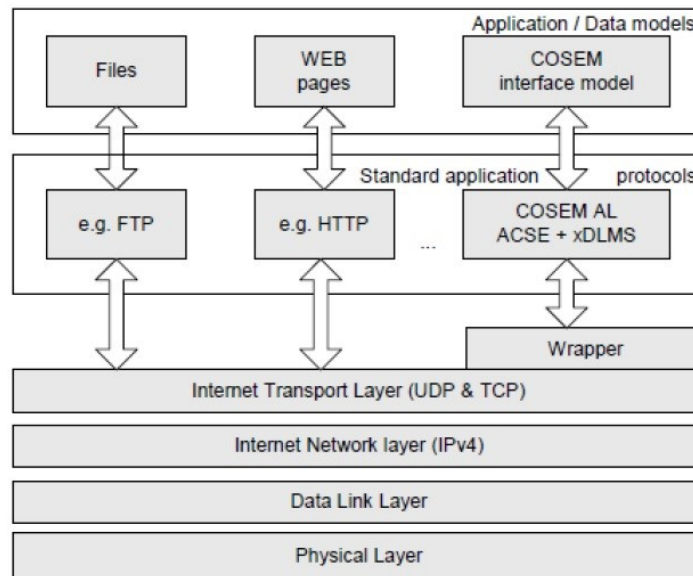


Figure 4: COSEM over IPv4 [10].

3.1.7 DLMS/COSEM security

3.1.7.1 Authentication

The authentication process involves establishing the identity of the communication partners before data communication services are requested or provided [5].

Three levels of authentication are available with different levels of authentication security as follows:

- **No security (Lowest Level Security)**

At the lowest security level, direct access to the data on the server is allowed. The goal of this level is to enable the client to obtain some basic information from the server. Therefore, no authentication mechanism is required, and the client can access the attributes and methods of the COSEM object [34], [35].

- **Low-Level Security (LLS)**

At this level, the server requires the client to authenticate itself by providing a password that is recognized by the server. If the password checked by the server is correct, the association is considered established [34], [35].

- **High-Level Security (HLS)**

At this level, both the server and the client must be authenticated to establish AA (Application Association). Various authentication mechanisms are available in HLS itself, including MD5², SHA-1³, GMAC⁴ used in this project, SHA-256⁵, and ECDSA⁶ [34], [35].

Table 1 shows the authentication mechanisms of DLMS/COSEM:

Authentication mechanism Names	Mechanism id
No Security	0
Low Level Security	1
High Level Security(HLS)	2
High Level Security using MD5	3
High Level Security using SHA-1	4
High Level Security using GMAC	5
High Level Security using SHA-256	6
High Level Security using ECDSA	7

Table 2: Authentication mechanisms of DLMS/COSEM [34].

During the application association (AA), the authentication mechanism ID is specified in Table 1. To achieve application association, the authentication mechanism is selected based on the ID of the mechanism, and the associated procedures are followed. The mechanisms ID 3 to 7 are considered as the mechanisms for authenticating both the client and the server [34].

In the server, the access rights to the objects of the classes are assigned according to the authentication mechanism [34].

In the HLS authentication process, four steps must be taken place as follows [36]:

Step1: Depending on the authentication mechanism chosen, the client passes a challenge (CtoS)⁷ to the server [36].

Step2: The server passes a challenge (StoC)⁸ to the client [36].

Step3: The challenge received from the server is processed by the client for the given application association, and the response is sent from the client to the server. The response is checked by the server and if it is correct, the authentication of the client is accepted by the server [36].

² The MD5 message-digest algorithm.

³ The Secure Hash Algorithm 1.

⁴ The Galois Message Authentication Code.

⁵ SHA-256 is used for the most popular authentication and encryption protocols.

⁶ Elliptic Curve Digital Signature Algorithm is a cryptographic algorithm.

⁷ “C” stands for the client, and “S” stand for the server, which means client to server.

⁸ Server to client.

Step4: The challenge received from the client is processed by the server for the given application association and the response is sent from the server to the client. The response is checked by the client and if it is correct, the authentication of the server is accepted by the client [36].

After the successful establishment of the application association (AA) between the client and the server, depending on the access right of the objects, the server processes the service request of the client, and the response is sent from the server to the client. Depending on the security context chosen during the data exchange, the data of the challenge-response mechanism is protected [10], [36].

3.1.7.2 Cryptography

To generate the response to the challenge issued by the client and server, the HLS authentication level requires the processing of cryptographic algorithms. Cryptography is a form of mathematics that deals with the transformation of information. Cryptography consists of two fundamental elements: an algorithm (cryptography method) and a key. Algorithms are mathematical functions and keys are used as parameters in transformations [10], [37].

DLMS/COSEM has a security suite with a collection of cryptographic algorithms as shown in Table 2. This security suite typically relies on symmetric key cryptographic communication, however, high-level security requires digital signatures and public key cryptographic communication [37].

Security Suite Id	Security suite name	Authenticated encryption	Digital signature	Key agreement	Hash	Key-transport	Compression
0	AES-GCM-128	AES-GCM-128	–	–	–	AES-128 key wrap	–
1	ECDH-ECDSA-AES-GCM-128-SHA-256	AES-GCM-128	ECDSA with P-256	ECDH with P-256	SHA-256	AES-128 key wrap	V.44
2	ECDH-ECDSA-AES-GCM-256-SHA-384	AES-GCM-256	ECDSA with P-384	ECDH with P-384	SHA-384	AES-256 key wrap	V.44
All other reserved	–	–	–	–	–	–	–

Table 3: DLMS/COSEM security suite [37].

According to Table 2, COSEM/DLMS symmetric key security system utilizes Gallois Counter Mode (GCM) in conjunction with the “AES-128”⁹ algorithm [37].

3.1.7.2.1 AES-GCM-128 algorithm

To secure the connection between the client and the meter, the most popular solution is the “AES-GCM-128” Security Suite 0 [38] that is used in this work.

Symmetric cryptography is used in Security Suite 0. In symmetric cryptography, the data sent is encrypted using GMAC ciphering, which is a variant of the GCM [3], [37].

⁹ AES(Advanced Encryption System) key length is 128 bits.

To encrypt the data, the following information must also be provided. Additionally, the following information can be used for decrypting the data as well.

- **Security level**

During the establishment of the connection, the client determines how the data is encrypted. There are three levels of security for data transfer [10], [38]:

- Authentication only
- Encryption only
- Authentication and Encryption level, which is used in this work.

- **System title**

This is an eight-octet value identifying the meter. System titles begin with the three-letter manufacturer ID (Flag ID). A serial number is then inserted at the end of the system title [38]. As shown in Figures C. 11, and C. 12 of Appendix C, the system title of the meter (server-side), is used to encrypt the data sent from the meter, and the system title of the software (client-side), is used to encrypt the data sent from the software.

- **Block cipher key (Encryption Key)**

Block cipher keys are 16 octets long values that specify the secret encryption key. To obtain this information, the meter manufacturer must provide a key that is different for each manufacturer [38].

- **Authentication key**

Authentication keys consist of 16 octets and are used in Additional Authentication Data (AAD). It is the responsibility of the meter manufacturer to provide this information, and each manufacturer has a unique key [38].

- **Invocation counter**

Each packet sent is distinguished by the Invocation Counter. When a new packet is sent, the Invocation Counter increases. Whenever the same packet is sent twice, the encrypted data differs due to the invocation counter [38].

Chapter 4: Project requirements

Project requirements can be summarized as follows:

4.1 On-premises data center

Considering that the main goal of this project is developing an on-premises IoT platform to read and manage smart gas meters, it is expected to have an on-premises data center where the software is installed along with a gas meter, which has the ability to show the APDUs (Application Protocol Data Units) exchanged with the software as debug strings on a PC. These debug strings (packets) are part of the DLMS messages exchanged between the meter and the software. They are translated and shown in Appendix C.

Moreover, it is also necessary to configure the network infrastructure on the PC and the metering device to test the developed web-based software (web application) on the local server.

It is also necessary to implement a class diagram for the underlying database (SQL server) as well as a corresponding object model.

4.2 Physical meter device

The metering device is needed to test the developed software. As mentioned in Section 4.1, the network infrastructure is also required to be configured in the meter. COSEM physical devices must be uniquely identified by their network-IP addresses in the TCP-UDP/IP-based profiles. Also, an additional identifier is required for the identification of COSEM client AP and server AP. To distinguish between applications, TCP provides an additional addressing capability at the transport level, called port.

For each smart gas meter, three major internal components are needed: an energy source, a microcontroller, and a communications interface.

Gas meters require batteries as an energy source. Two types of batteries are needed, such as metrological and communication batteries, shown in order as "Battery Estimated Remaining Use_0" and "Battery Estimated Remaining Use_1" COSEM objects in Appendix B.4. Each "Battery estimated remaining use time" object displays how long it is expected to remain before the battery is exhausted. The initial capacity of the battery energy (Battery Initial Capacity_N), which is expressed in mAh (milliampere-hours), and the total autonomy time specified by the manufacturer are used to calculate the remaining time [39].

A Digital-to-Analog Converter (DAC) and an Analog-to-Digital Converter (ADC) are typically included in a Microcontroller Unit (MCU) as one of the main components in the gas meter. Also, a GSM/GPRS module is required as a wireless communication interface to enable the meter to communicate with the rest of the grid [40]. As the result, the microcontroller in the smart gas meter must be able to communicate with the GSM module and LCD of the metering device [41].

The metering device requires an LCD that is used to display the current gas consumption and other necessary information such as the batteries' situation of the meter.

An optical port is also required in the case of meter reading with HDLC-based communication profiles (refers to Section 3.1.6.1), to permit local access during the installation or maintenance of the meter.

As the main source of energy in the gas meter, the "Metrological" battery powers the "Microcontroller", "LCD", and "Optical Port". Furthermore, the GSM module is charged by the "Communication" battery [39], [40].

4.2.1 Meter device parameters

The following manufacturer-specific parameters are required for each smart meter and must be set before the connection is started [38], as described in detail in Chapter 5 and Appendix B.2.

- Logical name or short name used.
- Server address.
- Client address.
- Interface type.
- Authentication type

4.3 GSM/GPRS based communication network

As mentioned in Section 4.2, a GSM/GPRS communication module is required for the transmission of direct read information. As the main feature of the smart meter, a SIM card is also required to integrate with the GSM modem.

4.4 Symmetric cryptography information

Since "AES-GCM -128" is used in this project along with "Security Suite 0", "Symmetric Cryptography" is also employed as described in detail in Section 3.1.7.2.1. Thus, when symmetric cryptography is used to encrypt and decrypt data, the following requirements need to be met [38].

- Used security level (AuthenticationEncryption security level is used in this work)
- System titles (for the software and the meter)
- Block cipher key (Encryption Key)
- Authentication key
- Invocation Counter

4.5 Development environment

Since DLMS/COSEM specifies an object-oriented data model, the C# programming language is suitable for use as an object-oriented programming language. As a result, Microsoft Visual Studio should be used as an integrated development environment (IDE) to develop the web application and access it through a web browser. Moreover, the ASP.NET Core framework is

well suited for developing the web application in a simplified programming model. The IIS (Internet Information Services) web server is also required to serve the HTML (HyperText Markup Language) pages or files requested by the client.

4.6 Documentation

The documents such as UNI/TS, blue and green books are required. UNI/TS book [39] contains the Italian regulation that defines a list of clients (public clients, management clients, etc., defined in Section 1.2), which are allowed to authenticate during communications between the software and the gas meter. Moreover, blue and green books specify the COSEM objects and their specifications, OBIS codes, interface classes, and other details [2], [10].

4.7 Gurux libraries

The Gurux libraries such as “Gurux.Common”, “Gurux.Net”, and “Gurux.DLMS” are required to use. For example, the “Gurux.DLMS” library is used as a high-performance component for reading the DLMS/COSEM gas meter.

4.8 Gurux GXDLMSDirector

It is necessary to use the “Gurux DLMS Translator” part of “Gurux GXDLMSDirector” in order to gain a better understanding of the DLMS protocol refer to Appendix C. It is possible to convert DLMS PDU or ASN.1 files to XML using Gurux DLMS Translator. Also, XML can be converted to PDUs or ASN.1 bytes using the mentioned translator.

4.9 Clear Terminal (CT)

A clear terminal (CT) is required for the AT commands to be set. AT commands are used to control the GSM modem on the meter. The figure below shows the AT commands on the clear terminal.

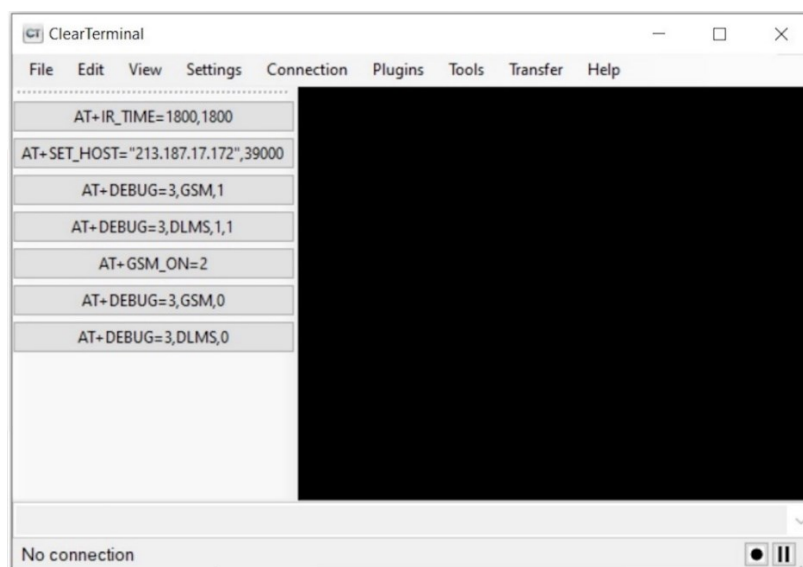


Figure 5: AT Commands included in the clear terminal.

Chapter 5: Project implementation

5.1 Connect the meter to the software by opening the TCP/IP port

If the meter acts as a server and the developed software connects to it, the meter must have a static IP address to be read via GPRS. Otherwise, the developed software acts as a server and it must have a static IP address, as is the case with this work.

The meter is set to connect to the server via a TCP/IP connection at a specified time interval. To accomplish this, a TCP/IP server must first be configured to listen on the specified port. Thus, from the "GXNet" class, an object named "server" is defined, which contains the "TCP Network Type" and the "Listening Port". After connecting the meter to the server, the events illustrated in Appendix B.1 are executed. Then, the IP and port of the connected meter are assigned to a new "GXNet" object, which can be used to read the meter information (IP and port), as described in detail in Appendix B.1.

5.2 Connect to the meter with "No Security" or Lowest-Level Security

Once a TCP/IP connection has been established between the meter and the software, a DLMS connection must also be established to exchange data. In DLMS/COSEM connection, the meter acts as a DLMS server and the software acts as a DLMS client. The DLMS connection is started with a "No Security" set up to read the "Logical Device Name". But, before starting the connection, by defining a new *client* object from the *GXDLMSSecureClient* class, the following manufacturer-specific device parameters need to be set as shown in Appendix B.2:

- Logical name or short name
- Server address
- Client address
- Interface type
- Authentication mechanism

When authentication (access security) is used, the server (meter) can grant different rights to the client. Without authentication, only reading is permitted. The serial number of the meter, which is the "Logical Device Name", is read as shown also in Appendix B.2. Moreover, two objects such as "Invocation Counter" and "Metering Point ID" are read after "Logical Device Name" in the "No Security" level. The "Invocation Counter" is specific for each device, and it is used when the software connects to the meter with "High-Level-Security".

The meter has been read in the "No Security" level to be recognized in the database by its "Logical Device Name", which has been obtained. After this step, the target is to read the meter with the "High-Level-Security". For this type of reading, the "Authentication Key" and "Block Cipher Key" (Encryption Key) are required. As a result, the database must first be read to determine the keys of the meter that have been identified by its "Logical Device Name".

The process of reading the database and finding the necessary keys (Authentication key and Block cipher key) is also illustrated in Appendix B.2 and its Figure B.2. 1, Table B.2. 1, and Table B.2. 2.

Then, it is necessary to close the connection and wait about 2 seconds. Afterward, a secure connection can be established.

5.3 Connect to the meter with High-Level Security + AES 128 encryption algorithm, and read the “Logical Device Name” of the meter and invocation counter

At this level, the authentication mechanism is “HLS GMAC” or “HighGMAC”, which has id 5 (Refer to Table 1). This means that high-level authentication and the “AES 128” encryption algorithm are enabled, referred to as "GMAC" ciphering [38], as described in Section 3.1.7.2.1.

A new *client* object from the *GXDLMSecureClient* class is defined again, and the following device parameters need to be set as shown in Appendix B.3.

- Logical name or short name
- Server address
- Client address
- Interface type
- Authentication mechanism (HighGMAC is used in this work)

Data transfer security is used, in which each packet is encrypted with *GMAC*. So, the following properties must first be defined as shown in Appendix B.3.

- AuthenticationKey
- BlockCipherKey (Encryption Key)
- SystemTitle
- Security level (AuthenticationEncryption is used in this work)
- Invocation counter

The third level of data transfer security (AuthenticationEncryption security) is used, which involves both authentication and encryption of all messages (Refer to Appendix B.3 and Section 3.1.7.2.1). Also, the “Ciphering.InvocationCounter” is set to zero.

Afterward, the "InitializeConnection" method is executed and the *AARQRequest* is sent to the meter and the *AAREResponse* is received from the meter (refer to Appendix C in which all DLMS messages are described). Thus, the client requests the establishment of the LN_With_Ciphering application context (Refer to the application contexts described in Section 1.2), which uses the “High-Level-Security GMAC” authentication mechanism as shown in Appendix C.

The "Logical Device Name" is read again, and a *session* is defined in the database to store data after each connection. Appendix B.3 illustrates the "DLMS_Connection_Session" table of the database, as well as the OBIS code and interface class of the "Logical Device Name" as the first object.

5.4 Read other remaining objects (converted volume, metering point id, ...) from the meter

The meter information is derived from the meter as objects below, which are obtained based on their OBIS codes defined in the program code of the software, as well as the interface classes defined in the Blue Book. Appendix B.4 describes in detail how to obtain these objects.

- **Logical device name** as shown in the previous section and Appendix B.

COSEM logical devices are identified by a globally unique Logical Device Name [2].

- **Metering Point ID**

Measuring results from the metering points must be conveyed to the commercial operations by the gas meter [2].

- **Absolute Converter Volume**

It contains information about the values logged by a volume converter, which includes the volume of the undisturbed converter, the index deviation, and the value at base conditions. The data is collected at the end of each measurement [2].

- **Daily Load Profile**

Additionally, the value of the daily corrected gas volume of the consumer's meter is measured by the "Daily Load Profile" COSEM object [39]. The value is presented in a chart as shown in Section 6.3.2.

- **Current Diagnostic**

It contains diagnostic and alarm data that are collected from the meter [2]. All the data of the "Current Diagnostic" COSEM object are defined in the "UNI/TS 11291-12-2" [39]. The table below shows the "Current Diagnostic" list or (Alarms list):

Bit	Description	Activated from the event with ID	Restore or from the event with ID
B0	Clock synchronization failed	10	11,12
B1	Complete Metrological Event Log	26	2
B2	Metrological Event Log $\geq 90\%$	27	2
B3	Measurement algorithm in error	20	21
B4	General device error	22	23
	Serious software error	85	-
B5	Flow error: overflow detected	40	41
	Flow Error: Reverse Flow Detected	42	43
B6	Memory error	66	-
B7	Copy of the least significant bit of the UNI / TS Device register Status	-	-
B8	Battery level less than 10%	74	75
B9	Critical battery level	94	
B10	Tamper detected (Tamper)	80	95
B11	Summer-time active	86	81
B12	Valve closed due to leaks	102	87
	Invalid valve password	106	31
	Valve closed because it was impossible to establish one communication with the device for a configurable time	107	31
	Valve closed but leaks present	113	31
	The valve cannot be opened or closed	114	31
B13	Sync		
B14	Reserved	-	-
B15	Reserved	-	-

Table 4: CurrentDiagnostic (Alarms) list [39].

- **BatteryEstimatedRemainingUse_0** and **BatteryEstimatedRemainingUse_1**

These two COSEM objects contain information regarding the batteries (metrological and communication batteries) in the device meter [2] (Refer to Section 4.2).

- **MetrologicalEventLogbook** and **NonMetrologicalEventLogbook**

Metrology tamper events involve instances in which an anomaly in the metrology system has been discovered as a result of tampering [2]. All the data of the “MetrologicalEventLogbook” and “NonMetrologicalEventLogbook” COSEM objects are defined in the “UNI/TS 11291-11-2” [42].

Finally, in order to control the valve of the gas meter (open/close the valve), the following COSEM objects must be obtained [2], [39] (Refer to Appendices B.4 and C).

- **Clock**

After obtaining the “Clock” COSEM object, the date and time of the meter are read.

- **Single Action Schedule**

After obtaining this COSEM object, the “executed_script” and “execution_time” attributes are written (set). In the “executed_script” attribute, the “Logical Name” and the “Selector” values specify the script to be executed. The "Selector" has three states. In this work, state 1 (close the valve) and state 2 (open the valve) are used. Also, in the “execution_time” attribute, the “time” and “date” values specify when the script is to be executed.

- **Disconnect Control**

After obtaining this COSEM object, the “output_state” attribute is set. This attribute describes the condition of the valve connection on the meter. The "true" state indicates that the valve is closed, while the "false" state indicates that the valve is open.

5.5 Save the meter information in the database

The obtained objects from the meter must be recorded in the database (refer to Figure B.5.1 of Appendix B.5) based on the “DLMS_Connection_Sessions” table defined in Appendix B.3.

So, the “DLMS_Connection_Sessions” table is edited and updated after reading all the COSEM objects.

Then, a path is defined for saving the log file after the measurements are complete as shown in Appendix B.5.

5.6 Close the connection between the software and the meter

Finally, the DLMS connection, as well as the TCP connection between the software and the meter are closed as shown in Appendix B.6.

Chapter 6: Tests and deployments of the project

6.1 Log File Translation

According to Section 5.5 and Appendix B.5, After reading the meter, a log file containing all the commands representing the data exchange between the meter and the software is saved in the specified path. These commands are DLMS messages, which are translated by the message translator part of the “Gurux GXDLMSDirector” to observe and evaluate the communication between the meter and the software. The translations of the DLMS messages are shown and explained in detail in Appendix C. In fact, these translations are demonstrations of all the DLMS messages used in the thesis. The DLMS messages in “Hex String” format are translated into messages in “XML” format.

6.2 Database Diagrams

The figure below shows the database diagrams containing thirteen tables such as:

- Userstbl- Usersessions- Usersessions
- DLMS_IoT_Devices
- DLMS_IoT_Device_Types
- DLMS_IoT_Device_Models
- DLMS_Keys
- DLMS_Customers
- DLMS_Object_Changes
- DLMS_Connection_Sessions- DLMS_Debug_Logs- DLMS_Data_Records- DLMS_Daily_Profiles

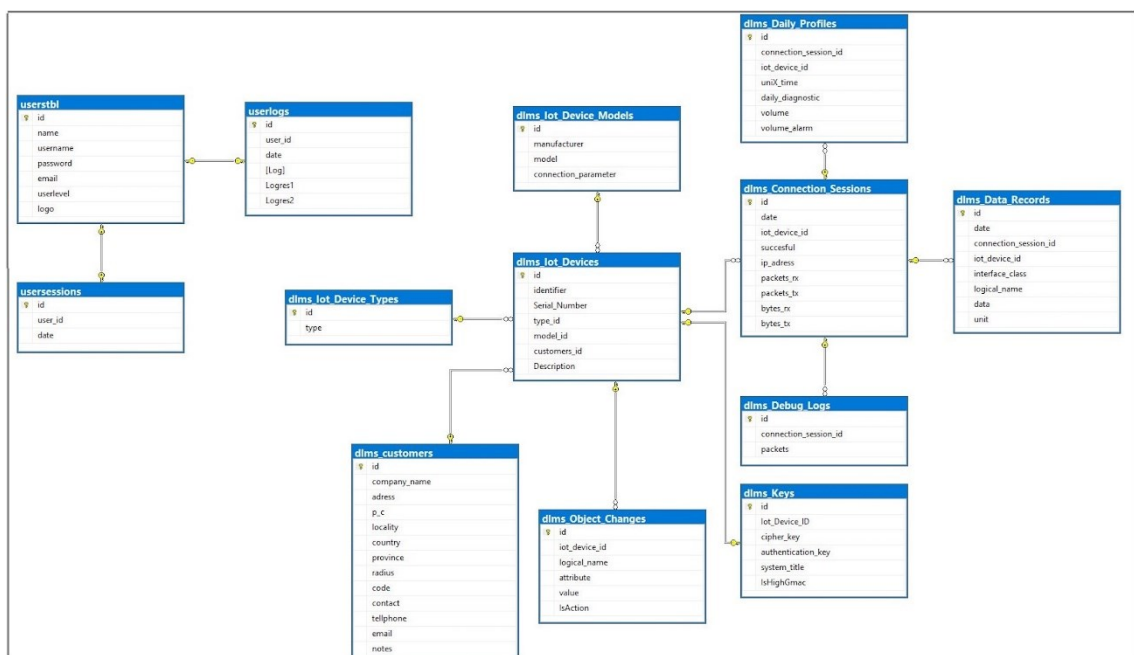


Figure 6: Database datagrams.

To show the datagrams in detail and separately, the following figure shows the diagrams associated with the users.

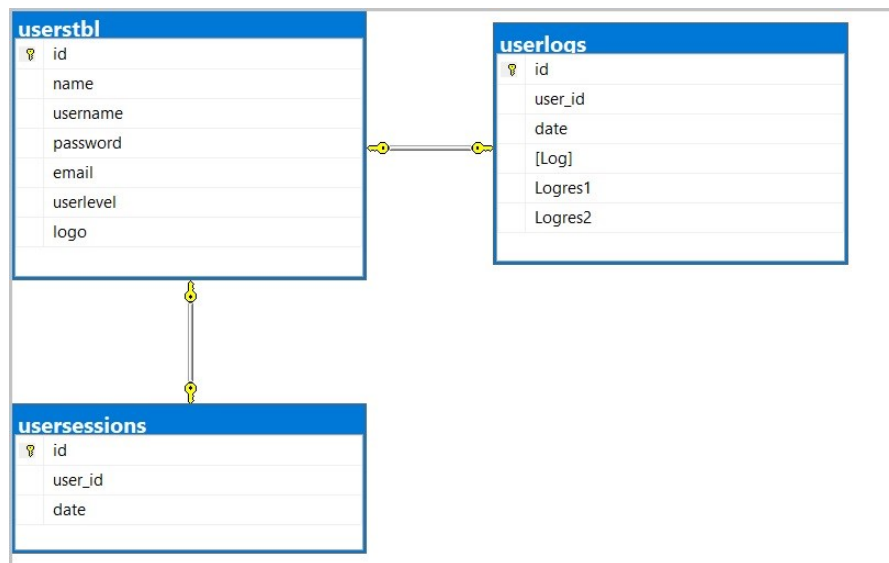


Figure 7: User's datagrams in database.

The following figure illustrates the diagrams associated with IoT devices.

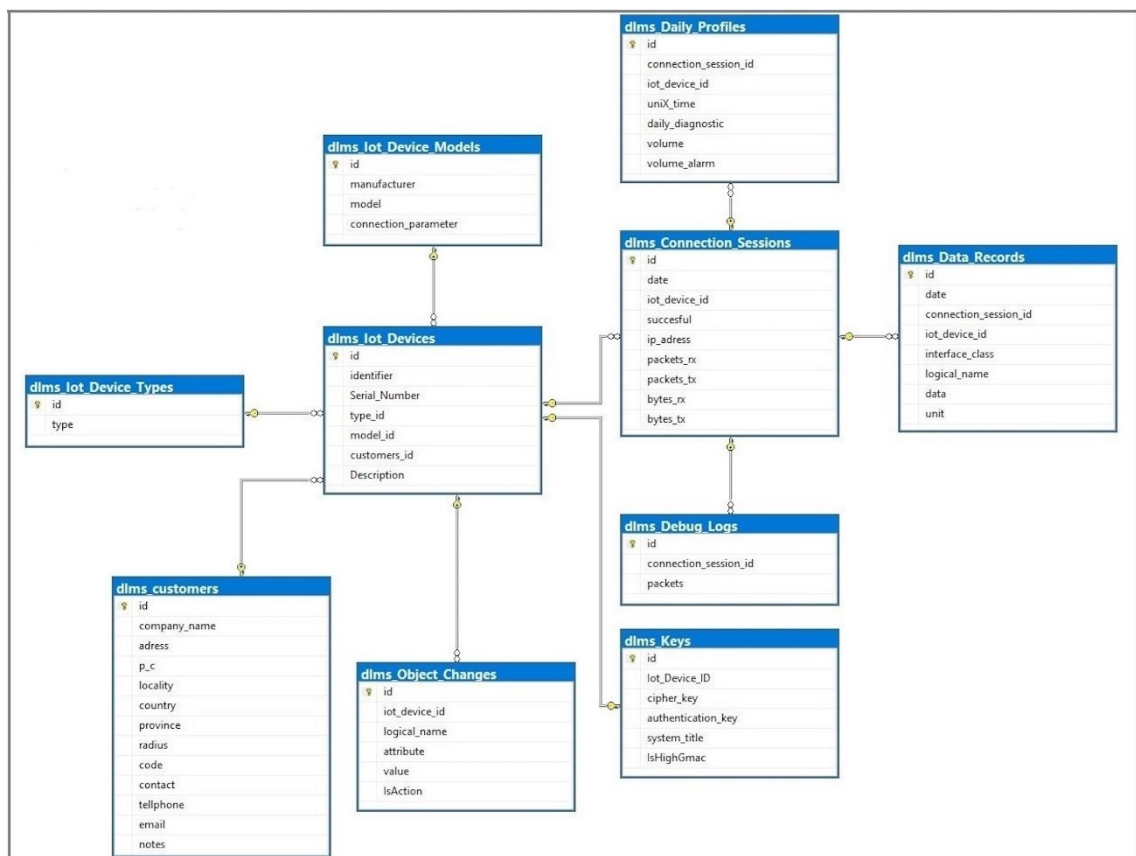


Figure 8: Diagrams related to the IoT_Devices diagram in the database.

6.3 Developed IoT Platform displays

According to Chapter 4 , Chapter 5 as well as Appendices A, B, and C, the IoT platform has been developed and tested using three different smart gas meters as figures below.



Figure 9: Three different smart gas meters were tested during the project.

The meter's display is active in the figure below, and the alarm status, battery status, and GSM status can be seen.



Figure 10: The meter's display.

The meter is set to "MU state" to establish the connection with the DLMS client (software).



Figure 11: Set the meter.

The following software pages have been developed:

6.3.1 Login Page

The figure below shows the “Login” page.

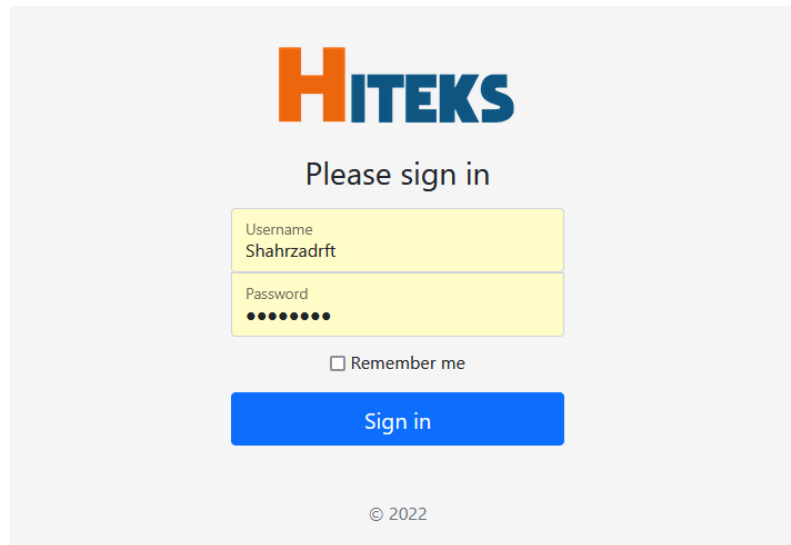
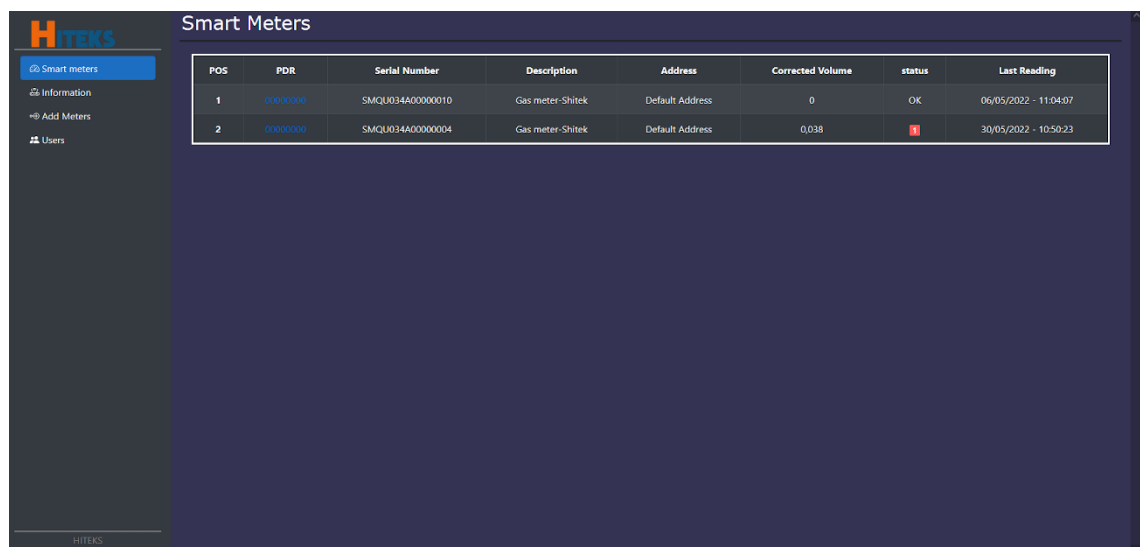


Figure 12: Login page.

6.3.2 Smart Meters Page

The figure below shows the “Smart Meters” page, which contains the list of the gas meters connected to the software, along with their “Serial Number”, “PDR” or customer code, which can be changed by the customer, “Description”, “Address” of the gas meter installation, “Corrected Volume” indicating the amount of gas that has entered the gas meter (Refer to Appendix B.4), “Status” displaying the status of alarms on a gas meter with a clickable notification, which displays the alarms on a separate page as Figure 14, and the “Last Reading” date and time associated with the “corrected volume” object.



POS	PDR	Serial Number	Description	Address	Corrected Volume	status	Last Reading
1	00000000	SMQL034A00000010	Gas meter-Shitek	Default Address	0	OK	05/05/2022 - 11:04:07
2	00000000	SMQL034A00000004	Gas meter-Shitek	Default Address	0,038	!	30/05/2022 - 10:50:23

Figure 13: Smart Meters page of the IoT platform.

- **Alarms Page**

The figure below illustrates the "Alarms" page, which can be accessed by clicking on the blinker notification on the "Smart Meters" page.

Alarms						
#	Devices	Code	Description	Priority	Status	Last Update
1	SMQU034A00000004	10	Tamper detected (Tamper)	High	-	30/05/2022 10:50:23

Figure 14: Alarms page of the IoT platform.

After clicking on each "PDR" or customer code on the "Smart Meters" page, the following page appears, which contains four sections: "General", "Read Data", "Daily Volume", "Graphics", and "Settings".

- **General Section**

The figure below illustrates the "General" section, which contains a summary of the installation information, gas volume information, battery status, and valve management, which includes the "open", "close" and "do not change" states. Moreover, the "General" part consists of a summary of the meter alarms in order of preference on the right side of the page.

The screenshot shows the 'General' section of the IoT platform interface. It features a sidebar with navigation options like 'Smart meters', 'Information', 'Add Meters', and 'Users'. The main content area is divided into several sections:

- Installation Info:** Includes fields for Serial Number (SMQU034A00000004), Name (Shitek), Last Contact (30/05/2022 10:50:23), and Address (Default Address).
- Counters:** Shows Tot.Corrected Volume (0.038) and Reading Date & Time (30/05/2022 10:50:23).
- Battery Status:** Displays Battery Life (402569600) and Battery Life 2 (221817600).
- Valve Management:** Shows Current Status (Open) and Next Connection (Do not change).

On the right side, there is an 'Alarms' table with the following data:

#	Devices	Code	Description	Priority	Status	Last Update
1	SMQU034A00000004	10	Tamper detected (tamper)	High	-	30/05/2022 10:50:23

Figure 15: General section of the IoT platform.

- **Read Data Section**

The figure below shows the “Read Data” section. Users can access the necessary information about the meter by clicking on the "Read Data" section. These data are COSEM objects obtained from the meter, such as:

- LDN (Logical Device Name) or Serial Number of the meter,
- Metering Point ID,
- Absolute Converter Volume,
- Battery Estimated Remaining Use 0 and 1.

Pos	Device	Description	Value	Unit	Last Update
1	SMQI034A00000004	Metering Point ID	00000000	None	30/05/2022 - 10:50:25
2	SMQI034A00000004	Absolute Converter Volume	0,038	CorrectedVolume	30/05/2022 - 10:50:28
4	SMQI034A00000004	Battery Estimated Remaining Use 0	442569600	Second	30/05/2022 - 10:50:32
5	SMQI034A00000004	Battery Estimated Remaining Use 1	221817600	Second	30/05/2022 - 10:50:35

Figure 16: Read Data section of the IoT platform.

- **Daily Volume Section**

The figure below depicts the "Daily Volume" section, which contains the daily information about the gas volume of the gas meter for each day.

Pos	Device	Date Time	diagnostic	Volume	volume under alarm
1	SMQI034A00000004	07/05/2022 06:00:00	0	17	0
2	SMQI034A00000004	08/05/2022 06:00:00	0	17	0
3	SMQI034A00000004	09/05/2022 06:00:00	0	17	0
4	SMQI034A00000004	10/05/2022 06:00:00	1024	36	0
5	SMQI034A00000004	11/05/2022 06:00:00	1024	36	0
6	SMQI034A00000004	12/05/2022 06:00:00	1024	36	0
7	SMQI034A00000004	13/05/2022 06:00:00	1024	36	0
8	SMQI034A00000004	14/05/2022 06:00:00	1024	36	0
9	SMQI034A00000004	15/05/2022 06:00:00	1024	38	0
10	SMQI034A00000004	16/05/2022 06:00:00	1024	38	0
11	SMQI034A00000004	17/05/2022 06:00:00	1024	38	0
12	SMQI034A00000004	18/05/2022 06:00:00	1024	38	0
13	SMQI034A00000004	19/05/2022 06:00:00	1024	38	0
14	SMQI034A00000004	20/05/2022 06:00:00	1024	38	0

Figure 17: Daily Volume section of the IoT platform.

- **Graphic Section**

As shown in the figure below, the "Graphics" section presents a chart of the daily gas volume of the gas meter regarding the previous section.



Figure 18: Graphics section of the IoT platform.

- **Settings Section**

The figure below depicts the "Setting" section, which contains general information about the customers.

The screenshot displays the 'Settings' section of the HITEKS IoT platform. It features a dark-themed interface with a sidebar on the left containing navigation options: 'Smart meters', 'Information', 'Add Meters', and 'Users'. The main content area shows a form titled 'Customer Information'. The form includes the following fields: 'Identifier' (text input), 'Description' (text input with value 'Gas meter Shilek'), 'Customer' (dropdown menu with value 'Default Address' and a 'Create...' button), 'Address' (text input), 'Province' (text input), 'Customer Code' (text input), and 'Telephone' (text input). A 'Save' button is located at the bottom left of the form.

Figure 19: Settings section of the IoT platform.

6.3.3 Information Page

In the following figure, all gas meters connected to the software are listed with detailed information, such as their serial numbers, identifiers, models, manufacturers, types, packet TX, packet RX, bytes RX, bytes TX, and last update.

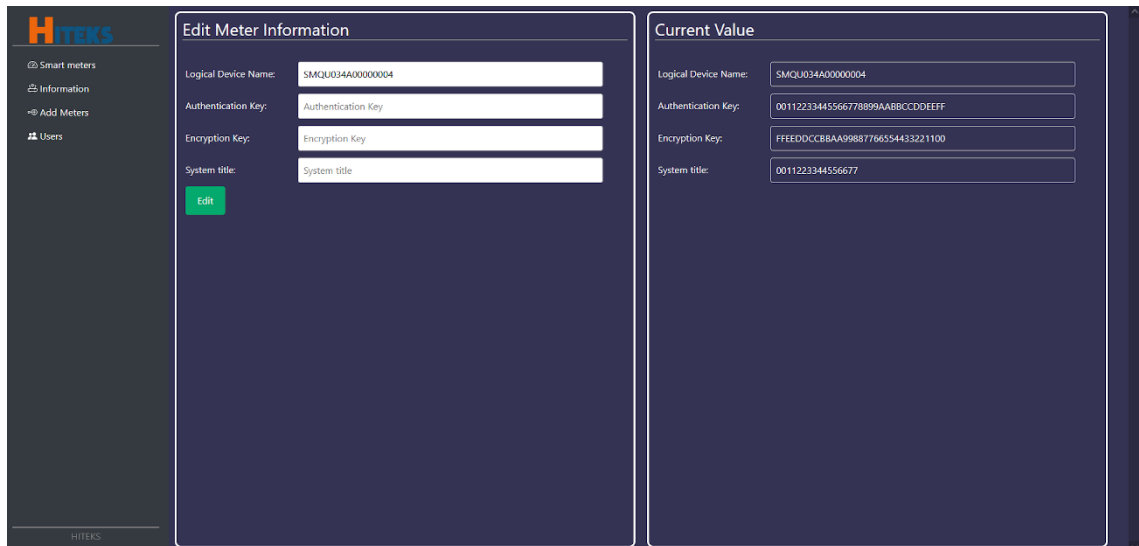


Figure 20: Information page of the IoT platform.

There are three buttons on this page for each meter: "Edit Meter", "Export Data", and "Delete Meter". By clicking on the "Delete Meter" button, the information of the meter is deleted. Also, the meter information can be changed by clicking on the "Edit Meter", as shown in Figure 21. Besides, the users can export the meters information as a "csv" file.

- **Edit Meter Section**

In the "Edit Meter" section, the user is allowed to modify the necessary specification of the meter, such as the "Authentication key", "Encryption key", and "System title".

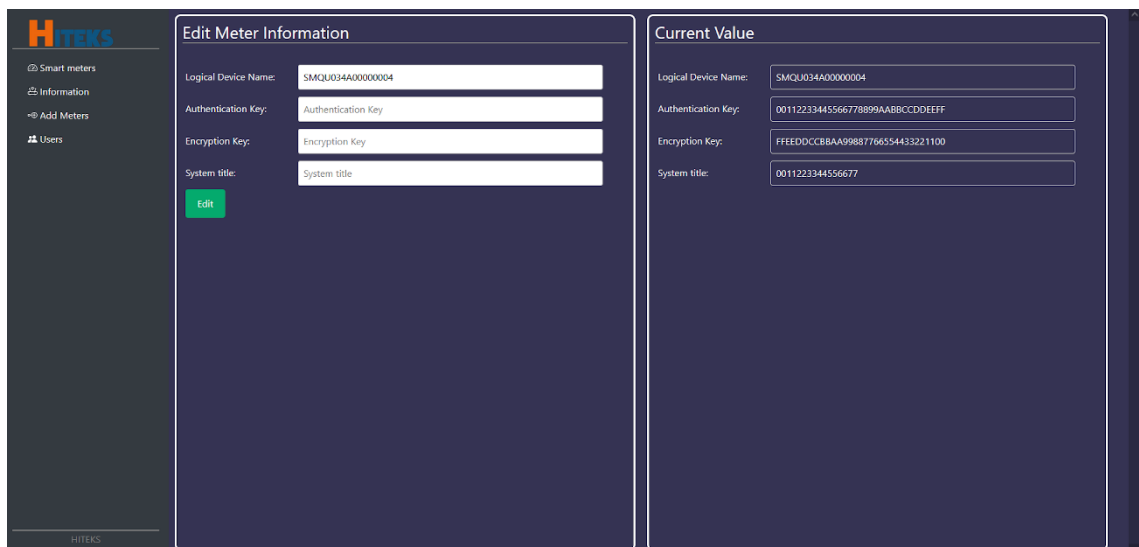


Figure 21: Edit Meter section of the IoT platform.

6.3.4 Add New Meter Page

There is a menu item in the software's menu bar called "Add Meters", where a user can enter the information for new meters, such as:

- Logical Device Name
- Authentication Key

- Encryption Key
- System title
- Authentication (Security Level that contains two levels of “Low-Level-Security” and “High-Level-Security”).

The “Add New Meter” page is shown as follows:

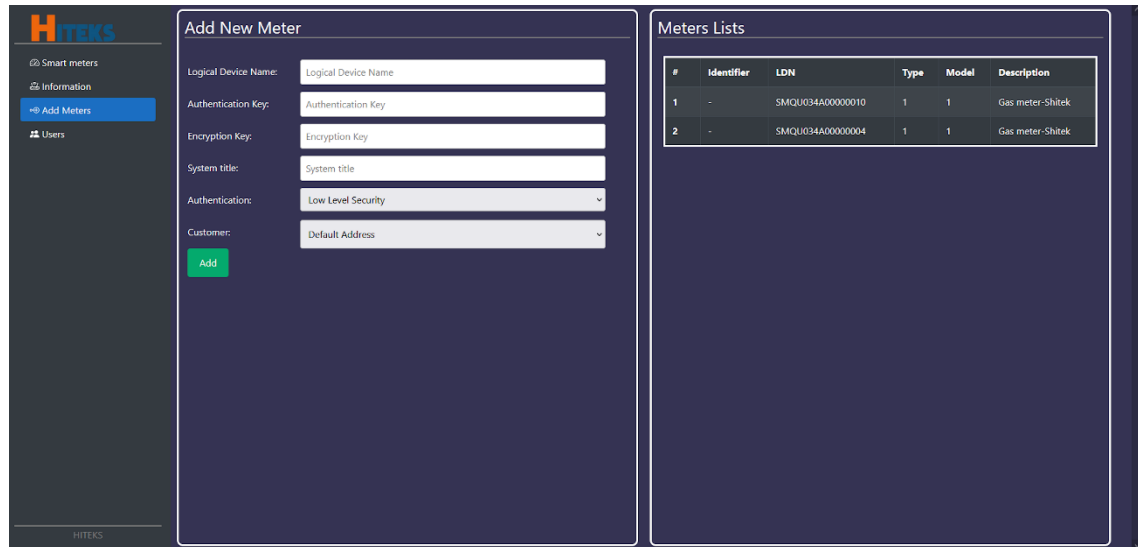


Figure 22: Add Meter page of the IoT platform.

Moreover, after entering a meter's information, the meter is added to a list of meters on the other side of the page along with its identifier, logical device name, type, model, and description.

Additionally, the required information must be added in the correct format. Otherwise, an error will be displayed. For instance, the authentication key must be 16 bytes long and in hexadecimal format. The cipher key must also be 16 bytes and in hexadecimal format. A system title must contain eight bytes and be in hexadecimal format.

6.3.5 Add New User Page

In the software's menu bar, there is an item titled "Users". On this page, new users can be added by providing the following information:

- Name
- Username
- Password
- Email
- User Level (It can be either user or admin)
- Avatar (A logo can be uploaded as user a photo)

This information is added to the list of users on the other side of the page, along with the user's ID, logo, username, email address, and level. Whenever the user's level is set to "User" the “Level” part of the users list will be false, but when the user's level is set to "Admin" the “Level” part of the users list will be true. This page is shown in the figure below:

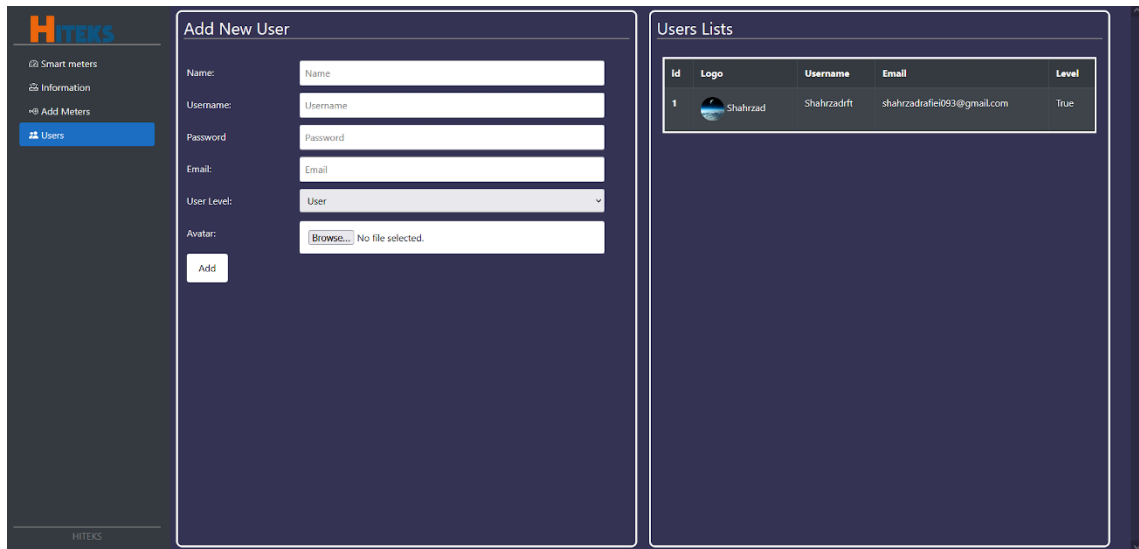


Figure 23: Users page of the IoT platform.

Users' passwords are encrypted using “AES-128” cryptography and stored in the database in “Hex string”¹⁰ format, as shown in the figure below:

	id	name	username	password	email	userlevel	logo
1	5	shahrzad	Shahrzadrft	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	1	U_20223_439089411.jpg
2	6	shahrzad2	Shahrzadrft2	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	0	NULL
3	7	Shahrzad3	Shahrzadrft3	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	1	NULL
4	8	Shahrzad4	Shahrzadrft4	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	0	U_20223_1188744701.jpg
5	9	shahrzad5	Shahrzadrft5	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	0	U_20223_1129819491.jpg
6	10	Shahrzad6	Shahrzadrft6	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	1	NULL
7	11	Shahrzad7	Shahrzadrft7	AA5F4DF671616318C31BD837522C8FC0	shahrzadrafiei093@gmail.com	1	U_20223_1840419651.jpg

Table 5: Usertbl-Users table of database.

For logging into the page mentioned in Section 6.3.1, the password is also decrypted using “AES-128”.

¹⁰ This is the binary value of the string in hexadecimal format.

Chapter 7: Conclusion

This thesis presents the development of an IoT platform, which has been installed and tested on the client's premises for the purpose of managing the smart gas meter located in the end consumer's field. Clients who have the software installed on their premises may include utilities and gas providers, while the end-users who have meters installed in their fields are the gas consumers. Since the developed web-based software (web application) is equipped with a data notification system, utilities will be notified of any alarms and events associated with consumers' gas meters. Moreover, the utilities are able to control (open/close) the valve of the gas meter installed in the consumers' fields using the developed software. Additionally, since the software has the capability to read multiple meters simultaneously, it will enhance interoperability between meters.

Based on the client/server model used in this work, the client (software) has received all the required COSEM objects from the server (meter). The collected data can be used to achieve the goal, which is to evaluate the gas consumption of consumers, leading to optimal gas consumption.

Based on the worldwide DLMS/COSEM standard for automatic meter reading, the data was transmitted wirelessly to the client using a GSM module. In reviewing the DLMS/COSEM protocol specification, it has been concluded that it would be beneficial to demonstrate communication with the meters using a standard programming language.

As a result of implementing the IoT platform, it is possible to communicate with any DLMS-compatible meter and retrieve basic readings such as "Logical Device Name", "Metering point ID", "Corrected volume", "Daily Corrected volume", "Current diagnostics", "Battery estimated remaining", "Valve Status", and "Last Connections" information. Thus, it is advisable to encourage the local development of software packages.

Considering that smart meters contain sensitive data, it is essential to implement security mechanisms, which have been supported in this work by the COSEM/DLMS security system for identifying both the client and the server. Therefore, a major reason for recommending the use of the DLMS/COSEM communication protocol for smart metering is the security services of this protocol.

Appendix A: Interface classes with their *class_id*

As illustrated in the following figures, the interface classes in each group are presented by increasing their *class_id*. The new interface classes are inserted at the end of the clauses defining the various groups of interface classes [2].

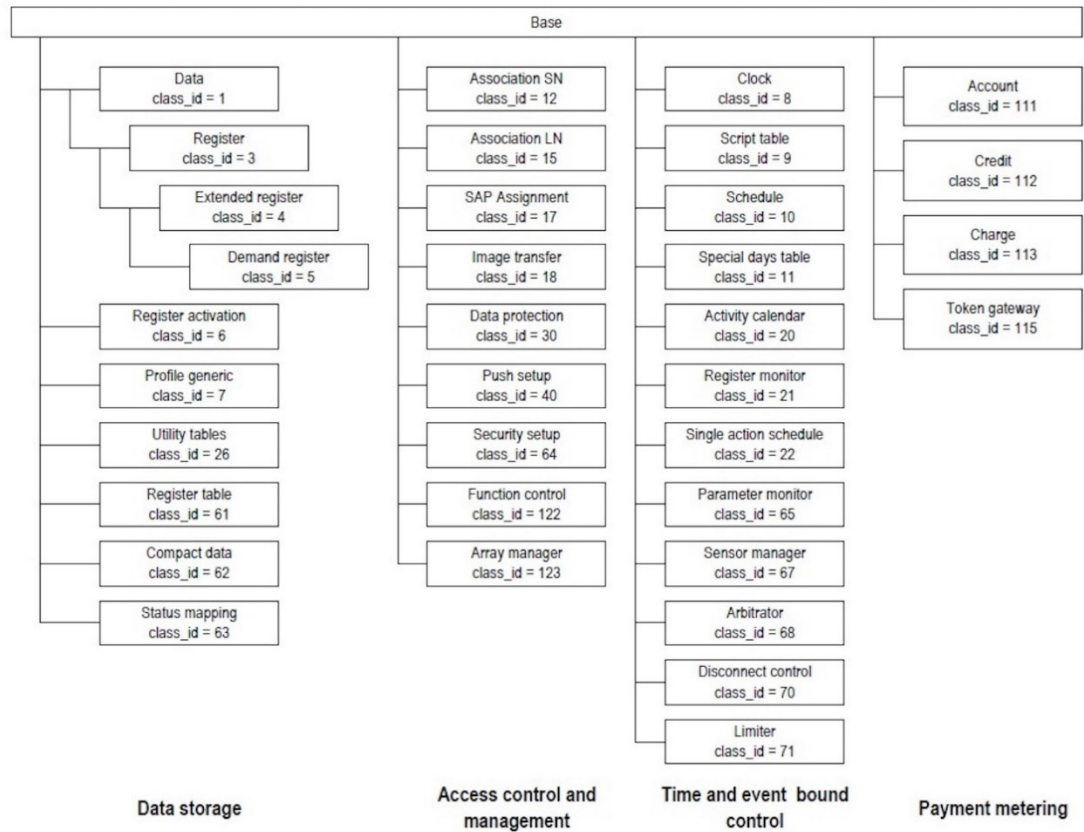


Figure A. 1: Overview of the Interface classes-Part1 [2].

For example, the "Register" Interface Class (IC) with *class_id*=3 represents the behaviour of a generic register, which contains measured or static information. Based on the structure of an interface class as shown in Table 1 in Section 3.1.4, the first attribute of the "Register" interface class is "Logical_Name" containing the OBIS identifier that identifies the contents of the "Register" class. The second attribute of the "Register" class is "Value", which contains the actual content of the "Register". Moreover, the third attribute of the "Register" class is "Scaler_unit", in which the scaler and the unit of the value are provided.

So, the attributes of the class "Register" with *Class_id*=3 are specified completely in the following three points [2], [5]:

- The first attribute is *logical_name* with a data type of *octet-string* and an *x* short name.
- The second attribute consists of a *value* with a dynamic data type of *Choice* and the short name of *x + 0x08*.
- The third attribute is *Scaler_unit* with a data type of *scal_unit_type* and the short name of *x + 0x10*.

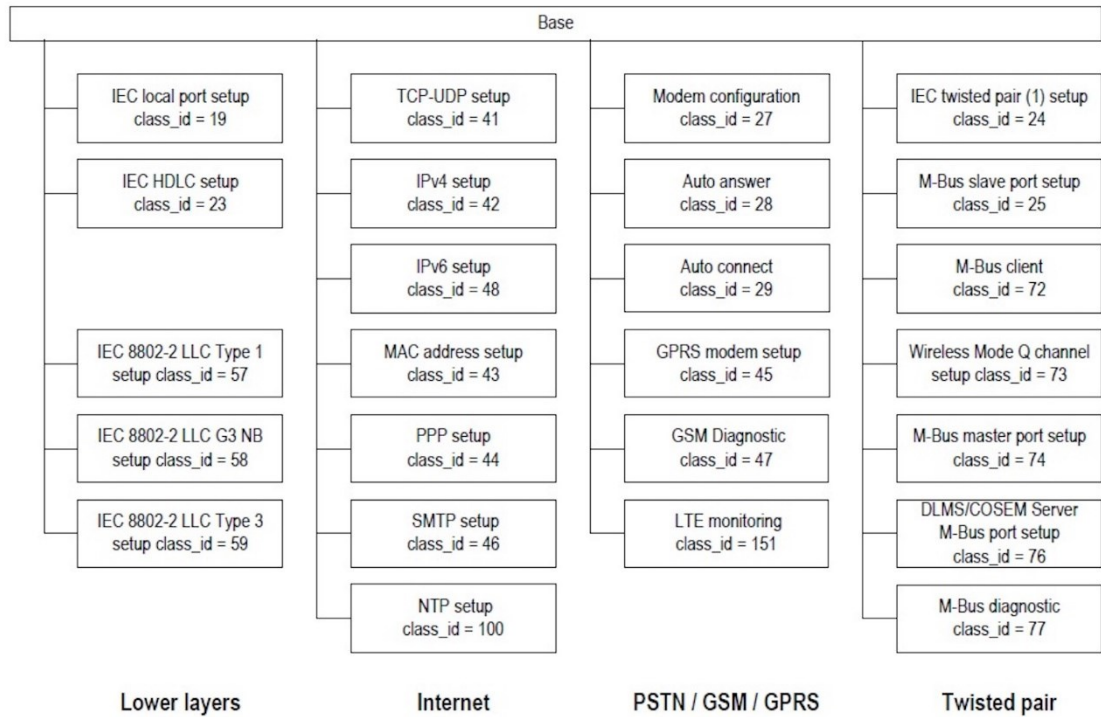


Figure A. 2: Overview of the Interface classes-Part2 [2].

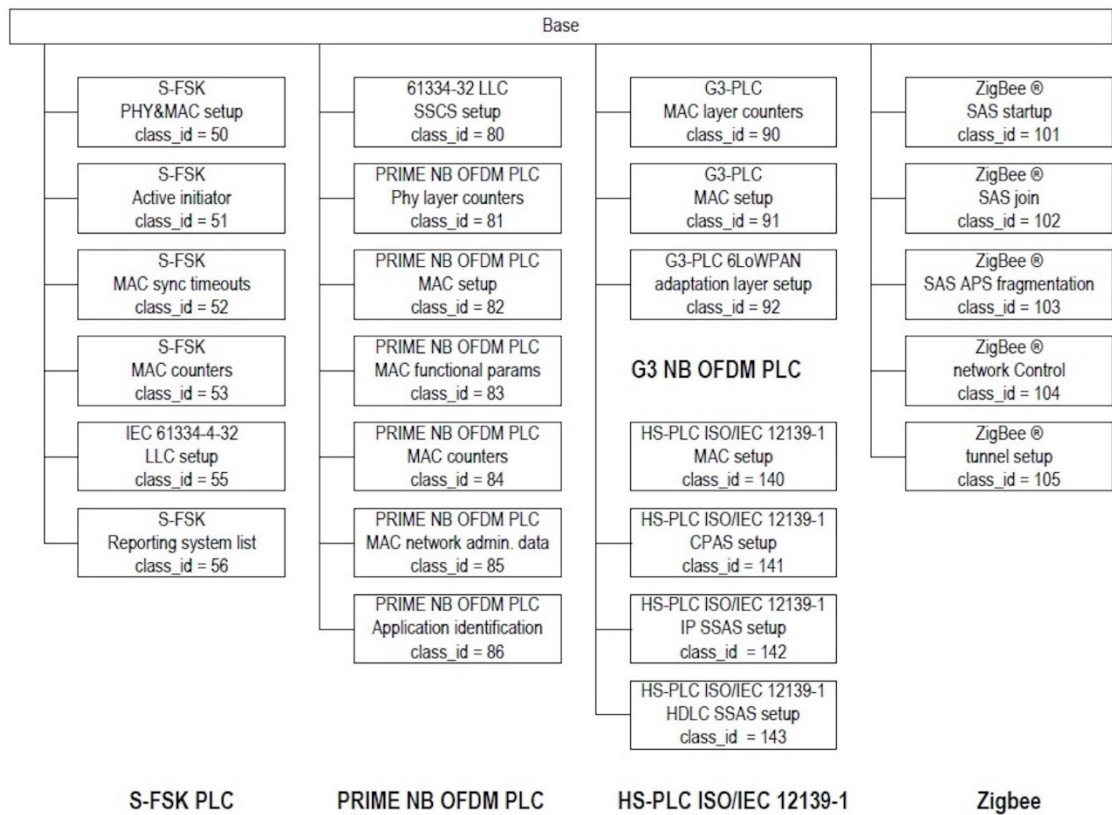


Figure A. 3: Overview of the Interface classes-Part3 [2].

Appendix B: Codes related to the project implementation

B.1 Connect the meter to the software

The codes below show the TCP/IP server, which is created from the *GXNet* class, as well as the process of connecting the meter to the software. A “Host Name” and a “Port Number” are configured, and then the port is opened to listen to the incoming connections from the meter.

Moreover, the events below are defined in the program:

- *OnClientConnected*, which is executed when a client (meter) is connected to the platform.
- *OnClientDisconnected*, which is executed when a client (meter) is disconnected from the platform.
- *OnReceived*, which is executed when data is received from the client (meter).

```
public void OnGet()
{
    DbRefresh(); // this is for read database.
    try
    {
        GXNet server = new GXNet(NetworkType.Tcp, 777);
        server.HostName = "213.187.17.172";
        server.Port = 39000;
        server.Open();
        server.OnClientConnected += OnClientConnected;
        server.OnClientDisconnected += OnClientDisconnected;
        server.OnReceived += OnReceived;
    }
    catch
    {
    }
}
```

According to the codes below, the meter is connected to the TCP/IP server, so the “OnClientConnected” event is executed, in which the “sender” object is the first input parameter, and the “e” event argument is the second input parameter. Here, the IP and the port of the connected meter are set as “e.Info” event argument, which is attached into the “sender” object. Then the “sender” object containing the meter information (IP and Port) is converted into a *GXNet* object and it is set equal to a new *GXNet* object named “server”. This “server” object containing attached meter information is also set equal to a new *GXNet* object named “cl”, which means “client”. The “cl” object can be used to read the IP and port of the connected meters during the project.

```
public void OnClientConnected(object sender, Gurux.Common.ConnectionEventArgs e)
{
    packetrx = 0;
    packettx = 0;
    for (int i = 0; i < 50; i++)
    {
        if (Lists.msgs[i] != null)
        {
```

```

        Lists.msgs[i] = null;
    }
}
GXDLMSReader.msgprint("client : { " + e.Info.ToString() + " } Is connected");
GXNet server = (GXNet)sender;
try
{
    using (GXNet c1 = server.Attach(e.Info))
    {
        server.Server = false;
        ReadMeter(c1, e.Info);
    }
}
catch (Exception ex)
{
}
}

```

B.2 DLMS connection with a “No Security” or Lowest-Level Security

According to Section 5.2, the meter is read first without security to be recognized.

As shown in the following codes, a new object named *client* is defined from the *GXDLMSSecureClient* class and "true" is set for *useLogicalNameReferencing*, "0x10" is set for *clientAddress*, "1" is set for *serverAddress*, "none" is set for the authentication mechanism without string password and "Wrapper" is set for *InterfaceType*. All the authentication mechanisms of DLMS/COSEM were described in Section 3.1.7.1.

```

//No security - just for recognizing the meter
client = new GXDLMSSecureClient(true, 0x10, 1, Gurux.DLMS.Enums.Authentication.
None, null, Gurux.DLMS.Enums.InterfaceType.WRAPPER);

```

The parameters of the *client* object were defined based on the parameters of the *GXDLMSSecureClient* class as follows:

```

public GXDLMSSecureClient(bool useLogicalNameReferencing, int clientAddress,
int serverAddress, Authentication authentication, string password,
InterfaceType interfaceType);

```

After that, the connection is initialized and according to the following codes, the “DLMSObj” object from the *GXDLMSData* class is defined and the “**Logical Device Name**” COSEM object (serial number of the meter) is invoked from the *DLMSParameters* class that has been defined in the *Lists* class of the developed program. By using the OBIS code of the “Logical Device Name” COSEM object defined in the Blue Book, the *class_id* of this COSEM object can be identified, and by using the “Interface Classes” tables as shown in Appendix A, the “Interface Class” of this COSEM object, can be also determined, which is *Data*. Then the second attribute of the LDN COSEM object is read, which is *Value* (the value of the meter serial number). After that, the result is converted to a string.

```

reader = new GXDLMSReader(client, media, TraceLevel.Verbose);

```

Appendices

```
reader.InitializeConnection();
GXDLMSData DLMSObj = new GXDLMSData(Lists.DLMSParameters.LogicalDeviceName);
// IC (Interface Class) : Data -> blue book by name : Class ID 1

object result = reader.Read(DLMSObj, 2); //The second attribute of this object
is "value" = The value of the Meter Serial Number(LDN -> Logical Device Name)
```

According to the following codes, figures, and tables, the next step is to read the database with the "DB_read" method to obtain the "Authentication Key" and "Block Cipher Key" (Encryption Key) by using the LDN (Logical Device Name) COSEM object that has been acquired.

The input for the "DB_read" method is "strresult", which contains the LDN or serial number of the device, which has been captured and has been converted to a string. However, if the meter or DLMS keys are not recognized, the connection is terminated.

```
DB_read(strresult); // read database for authentication key and block cipher
key - strresult is logical device name
if (iotdev.Count == 0)
{
    // Errors - not recognize meter(no meters)
    reader.Close(); // disconnect meter
    media.Close(); //close the tcp connection
    return;
}
else if (dlmskeys.Count == 0)
{
    // Errors - not recognize meter(no encryption keys)
    reader.Close(); // disconnect meter
    media.Close(); //close the tcp connection
    return;
}
```

In the "DB_read" method, the following is defined:

```
public void DB_read(string LDN)
{
    // var context = (dlmsContext)_services.GetService(typeof(dlmsContext));
    using (var _context = new dlmsContext())
    {
        iotdev = _context.dlms_Iot_Devices
            .Where(x => x.Serial_Number == LDN)
            .Select(x => new Views.dlms_iot_devices_view
            {
                id = x.id,
                identifier = x.identifier,
                Serial_Number = x.Serial_Number,
                model_id = x.model_id,
                type_id = x.type_id,
                Description = x.Description,
            }).OrderBy(x => x.id).ToList();
        if (iotdev.Count >= 1)
    }
}
```

```

    {
        dlmskeys = _context.dlms_Keys
            .Where(x => x.Iot_Device_ID == iotdev[0].id)
            .Select(x => new Views.dlms_keys
                {
                    id = x.id,
                    Iot_Device_ID = x.Iot_Device_ID,
                    cipher_key = x.cipher_key,
                    authentication_key = x.authentication_key,
                    system_title = x.system_title
                }).OrderBy(x => x.id).ToList();
    }
}
}

```

The "DLMS_IoT_Devices" table is initially read from the database, then the "DLMS_keys" table is read to identify the specific meters' keys. As shown in Table B.2. 1, the "Cipher keys," the "Authentication keys," and the "System titles" of each device are stored in the "DLMS_keys_table" in "Varbinary" format. If "IshighGMAC" is set to 1, it means that the user's security level is high, while if it is set to zero, it signifies that the user's security level is low.

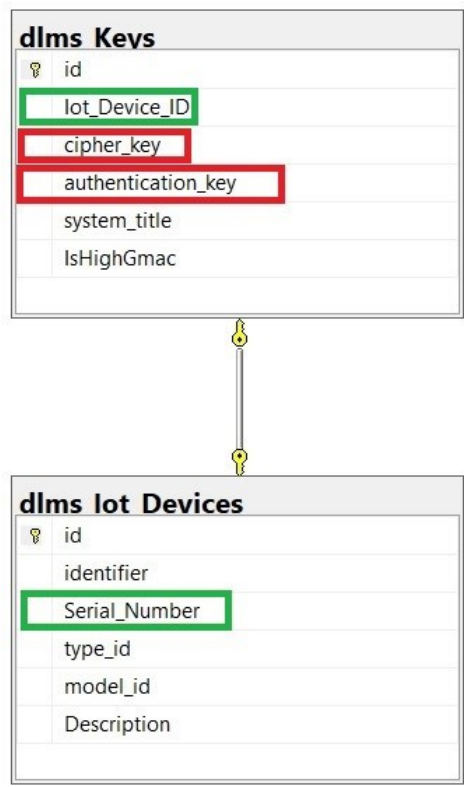


Figure B.2. 1: "DLMS_IoT_Devices" and "DLMS_Keys" diagrams.

	id	identifier	Serial_Number	type_id	model_id	Description
1	1	-	SMQU034A00000010	1	1	Gas meter-Shitek
2	3	-	SMQU034A00000004	1	1	Gas meter-Shitek
3	5	-	WTTS034020007307	1	1	-
4	6	-	WTTS034021014133	1	1	-
5	7	-	SMQU034A00000009	1	1	Gas meter-Shitek

Table B.2. 1: DLMS_IoT_Device table.

	id	lot_Device_ID	cipher_key	authentication_key	system_title	IsHighGmac
1	1	1	0xFFEEDDCCBBA99887766554433221100	0x00112233445566778899AABBCCDDEEFF	0x0011223344556677	1
2	3	3	0xFFEEDDCCBBA99887766554433221100	0x00112233445566778899AABBCCDDEEFF	0x0011223344556677	1
3	5	5	0x36FF80ABFA994D3897AF5A601D983621	0x36FF80ABFA994D3897AF5A601D983621	0x5341435341435341	1
4	6	6	0x47B381828E684380A634078B5C149876	0x47B381828E684380A634078B5C149876	0x5341435341435341	1
5	7	7	0xFFEEDDCCBBA99887766554433221100	0x00112233445566778899AABBCCDDEEFF	0x0011223344556677	1

Table B.2. 2: DLMS_Keys table.

At this level (No Security), two additional COSEM objects are obtained, namely "**Metering Point ID**" and "**Invocation Counter**" if the meters and their keys are recognized correctly. Then the connection is closed, and two seconds should be waited.

```
DLMSObj = new GXDLMSData(Lists.DLMSParameters.MeteringPointID);
DLMSObj = new GXDLMSData(Lists.DLMSParameters.InvocationCounter);
reader.Close(); // disconnect meter

Thread.Sleep(2000); // 2s delay
```

B.3 High Level Security & encryption

As shown in the following code, a new object named "client" is defined from the *GXDLMSSecureClient* class and "true" is set for *useLogicalNameReferencing*, "1" is set for *clientAddress*, "1" is set for *serverAddress*, "HighGMAC" is set for the authentication mechanism, "null" is set for string password, and "Wrapper" is set for *InterfaceType*.

```
//HLS HighGMAC -> High Level Security & encryption
client = new GXDLMSSecureClient(true, 1, 1, Gurux.DLMS.Enums.Authentication.
HighGMAC, null, Gurux.DLMS.Enums.InterfaceType.WRAPPER);
```

The following codes are the properties that must be set to use "GMAC" to encrypt each packet:

```
client.Ciphering.AuthenticationKey = dlmskeys[0].authentication_key;
client.Ciphering.BlockCipherKey = dlmskeys[0].cipher_key;
client.Ciphering.SystemTitle = dlmskeys[0].system_title;
```

```
client.Ciphering.Security = Gurux.DLMS.Enums.Security.AuthenticationEncryption;
// Application context name : LN_With_Ciphering
```

```
client.Ciphering.InvocationCounter = 0;
```

The two lines above show the use of both authentication and encryption (LN with Ciphering application context name) and the “Invocation Counter”, which is set to 0 as its value was zero during the "No security" meter reading step.

According to the following codes and figure, the connection is initialized, and the "Logical Device Name" COSEM object is invoked again using the *GXDLMSData* class, the OBIS codes defined in the "Lists" class of the Visual Studio program, as well as the other specifications defined in the Blue Book for this object. Then the second attribute of the LDN COSEM object is read, which is *Value* (the value of the meter serial number). After that, the result is converted to a string.

```
reader = new GXDLMSReader(client, media, TraceLevel.Verbose);
reader.InitializeConnection();
DLMSObj = new GXDLMSData(Lists.DLMSParameters.LogicalDeviceName); // IC : Data
-> blue book by name : Class ID 1
result = reader.Read(DLMSObj, 2); //Its second attribute is value = Meter
Serial Number(LDN -> Logical Device Name)
strresult = objtostr(result);
```

“Logical Device Name” OBIS Code in “Lists” class and “Blue Book”:

```
public const string LogicalDeviceName = "0.0.42.0.0.255";
```

COSEM logical device name object	IC	OBIS code					
		A	B	C	D	E	F
COSEM logical device name	1, Data ^a	0	0	42	0	0	255
^a If the IC "Data" is not available, "Register" (with scaler = 0, unit = 255) may be used.							

Table B.3. 1: COSEM logical device name object table [2].

First, a “session” is required to store each connection information as the following codes and figure. First, a session is written with false successful and when the reading is complete, the session successful becomes true (refer to Appendix B.5).

```
DB_write_session(strresult, false, ip, 0, 0, 0, 0);
```

The following codes are defined in the “DB_write_session” method:

```
public void DB_write_session(String LDN, bool succesful, string ip, int
packet_rx, int packet_tx, int byte_rx, int byte_tx)
{
    // var context = (dlmsContext)_services.GetService(typeof(dlmsContext));
    using (var _context = new dlmsContext())
    {
```

```

        var iot_d = _context.dlms_Iot_Devices.Where(a => a.Serial_Number ==
LDN).OrderBy(x => x.id).LastOrDefault();
        if (iot_d != null)
        {
            var connection_session = new dlms_connection_session(DateTime.Now,
iot_d.id, succesful, ip, packet_rx, packet_tx, byte_rx, byte_tx);

_context.dlms_Connection_Sessions.AddRange(connection_session);
_context.SaveChanges();
        }
        else { return; }
    }
}

```

dlms Connection Sessions	
id	
date	
iot_device_id	
succesful	
ip_adress	
packets_rx	
packets_tx	
bytes_rx	
bytes_tx	

Figure B.3. 1: DLMS_Connection_Sessions diagram in database.

B.4 Capturing the other required objects from the meter

As shown in the previous section, once the "Logical Device Name" COSEM object has been obtained, the remaining COSEM objects must be obtained using the same approach as the codes and figures below.

The "Metering Point ID" COSEM object is obtained using the *GXDLMSData* class, the OBIS codes defined in the "Lists" class of the Visual Studio program and the other specifications of this object defined in the Blue Book. Then the second attribute of the "Metering Point ID" COSEM object is read, which is *Value* (the value of the Metering Point ID). After that, the result must be converted into a "byte array" and then stored in the "DLMS_Data_Record" table as following codes.

```

DLMSObj = new GXDLMSData(Lists.DLMSParameters.MeteringPointID);
result = reader.Read(DLMSObj, 2);

string metering_string = objtostr(result);
DB_write_records(strresult, 1, strLNtoarray(DLMSObj.LogicalName), Encoding.
ASCII.GetBytes(metering_string), 0x0);

```

"MeteringPointID" OBIS Code in "List" class and "Blue Book":

```
public const string MeteringPointID = "0.0.96.1.10.255";
```

Metering point ID objects	IC	OBIS code					
		A	B	C	D	E	F
Metering point ID	1, Data ^a	0	b	96	1	10	255

^a If the IC "Data" is not available, "Register" or "Extended register" (with scaler = 0, unit = 255) may be used.

Table B.4. 1: Metering point ID object table [2].

The "Absolute Converter Volume" COSEM object is obtained using the *GXDLMSSData* class, the OBIS codes defined in the "Lists" class of the Visual Studio program and the other specifications of this object defined in the Blue Book. Then the second attribute of the "AbsoluteConverterVolume" COSEM object is read, which is *Value* (the value of the Absolute Converter Volume). Then its third attribute is read, which is the *Scalar* and *Unit* of the object, as shown in Appendix C. After that, the result must be converted into a "byte array" and then stored in the "DLMS_Data_Record" table as following codes.

```
GXDLMSSRegister regobj = new GXDLMSRegister(Lists.DLMSPParameters.
AbsoluteConverterVolume);
result = reader.Read(regobj, 2);
Object result2 = reader.Read(regobj, 3);

DB_write_records(strresult, 3, strLNtoarray(regobj.LogicalName),
Encoding.ASCII.GetBytes(c), Lists.unitTobyte(regobj.Unit.ToString()));
```

"AbsoluteConverterVolume" OBIS Code in "List" class and "Blue Book":

```
public const string AbsoluteConverterVolume = "7.0.13.2.0.255";
```

Name	Symbol	OBIS code
Indexes		
Forward absolute meter volume, index, at metering conditions	V_m	7.0.3.0.0.255
Forward absolute converter volume, index, at metering conditions	V_m	7.0.13.0.0.255
Forward absolute converter volume, index, corrected value	V_c	7.0.13.1.0.255
Forward absolute converter volume, index, at base conditions	V_b	7.0.13.2.0.255

Table B.4. 2: Absolute converter volume object table [2].

The obtained "Absolute Converter Volume" COSEM object is considered at base conditions. The volume at base conditions can be calculated [2] using the following equation:

$$V_b = C \times V$$

Where:

- V_b = Volume at base conditions,

Appendices

- $V =$ It can be either V_m or V_c that are volume at metering conditions or corrected volume.
- $C =$ The conversion factor provided by the relationship as follows:

$$C = (P / P_b) \times (T_b / T) \times (Z_b / Z)$$

Where Z represents the compressibility factor that considers the difference between the measured and ideal gas in terms of compression. This is a function of pressure and temperature as equation below:

$$Z = f(P, T)$$

In addition to this, the daily corrected gas volume of the meter is also captured and recorded in the database by reading the second and the third attributes of the “**Daily Load Profile**” COSEM object as described in Section 5.4 and on the Graphics page in Section 6.3.2.

The “**Current Diagnostic**” COSEM object is defined using the *GXDLMSData* class, the OBIS codes defined in the “Lists” class of the Visual Studio program and the other specifications of this object defined in the Blue Book. Then the second attribute of the “Current Diagnostic” COSEM object is read, which is *Value* (the value of the CurrentDiagnostic). After that, the result must be converted into a “byte array” and then stored in the “DLMS_Data_Record” table as following codes.

```
regobj = new GXDLMSRegister(Lists.DLMSPParameters.CurrentDiagnostic);
result = reader.Read(regobj, 2);

DB_write_records(strresult, 3, strLNtoarray(regobj.LogicalName), BitConverter.
GetBytes (Convert.ToInt32(regobj.Value)), 0x0);
```

“CurrentDiagnostic” OBIS Code in “List” class and “Blue Book”:

```
public const string CurrentDiagnostic = "7.0.96.5.1.255";
```

Data	class_id	Logical name	attribute_id	data_index	Size (bytes)	Type	Value
1	2	3	4	5	6	7	8
Template Id	62	0-0:66.0.0.255	4	0	1	unsigned	1
Current Diagnostic	3	7-0:96.5.1.255	2	0	2	long-unsigned	0x4200
Daily Diagnostic	3	7-1:96.5.1.255	2	0	2	long-unsigned	0x4108
Billing Period Diagnostic	1	7-2:96.5.1.255	2	0	2	long-unsigned	0x4308
Synchronization event counter	1	0-0:96.15.2.255	2	0	2	long-unsigned	763
Metrological firmware version	1	7-0:0.2.1.255	2	0	8	octet-string	"ABCDEFGH"
Metrological event counter	1	0-0:96.15.1.255	2	0	2	long-unsigned	1532
Non-metrological firmware version	1	7-1:0.2.1.255	2	0	8	octet-string	"DEFGHIJK"

Table B.4. 3: Current Diagnostic object table [2].

Appendices

The "Battery Estimated Remaining Use_0" and "Battery Estimated Remaining Use_1" COSEM objects are obtained using the *GXDLMSTData* class, the OBIS codes defined in the "Lists" class of the Visual Studio program and the other specifications of these objects defined in the Blue Book. Then their second attribute is read, which is *Value* (the value of the objects) and then their third attribute is read, which is the *Scalar* and *Unit*. After that, the results must be converted into a "byte array" and then stored in the "DLMS_Data_Record" table as the following codes.

```
regobj = new GXDLMSRegister(Lists.DLMSPParameters.BatteryEstimatedRemainingUse_0);
result = reader.Read(regobj, 2); //value
result2 = reader.Read(regobj, 3); //scalar, unit

DB_write_records(strresult, 3, strLNtoarray(regobj.LogicalName), Encoding.ASCII.GetBytes(c),
Lists.unitTobyte(regobj.Unit.ToString()));
```

```
regobj = new GXDLMSRegister(Lists.DLMSPParameters.BatteryEstimatedRemainingUse_1);
result = reader.Read(regobj, 2); //value
result2 = reader.Read(regobj, 3); // scalar, unit

DB_write_records(strresult, 3, strLNtoarray(regobj.LogicalName), Encoding.ASCII.GetBytes(c),
Lists.unitTobyte(regobj.Unit.ToString()));
```

"BatteryEstimatedRemainingUse_0" and "BatteryEstimatedRemainingUse_1" OBIS Codes in "List" class and "Blue Book":

```
public const string BatteryEstimatedRemainingUse_0 = "0.0.96.6.6.255";
public const string BatteryEstimatedRemainingUse_1 = "0.1.96.6.6.255";
```

General and service entry objects	OBIS code					
	A	B	C	D	E	F
Battery estimated remaining use time	0	b	96	6	6	
Aux. supply use time counter	0	b	96	6	10	
Aux. voltage (measured)	0	b	96	6	11	

Table B.4. 4: Battery estimated remaining object table [2].

The "Metrological Event Logbook" and "Non-Metrological Event Logbook" COSEM objects are obtained using the *GXDLMSTData* class, the OBIS codes defined in the "Lists" class of the Visual Studio program and the other specifications of these objects defined in the Blue Book. Then their second attribute is read, which is *Buffer* and then their third attribute is read, which is the *Capture_objects*.

```
GXDLMSPProfileGeneric prof_obj = new GXDLMSProfileGeneric(Lists.DLMSPParameters.
MetrologicalEventLogbook);
result = reader.Read(prof_obj, 2);
result2 = reader.Read(prof_obj, 3);
```

Appendices

```
prof_obj = new GXDLMSProfileGeneric(Lists.DLMSParameters.NonMetrologicalEventLogbook);
result2 = reader.Read(prof_obj, 3);
```

“MetrologicalEventLogbook” and “NonMetrologicalEventLogbook” OBIS Codes in “List” class and “Blue Book”:

```
public const string MetrologicalEventLogbook = "7.0.99.98.1.255";
public const string NonMetrologicalEventLogbook = "7.0.99.98.0.255";
```

Event log objects	IC	OBIS code					
		A	B	C	D	E	F
Event log	7, Profile generic	a	b	99	98	e	255 ^a
^a F = 255 means a wildcard here.							
NOTE 1 Event logs may capture for example the time of occurrence of the event, the event code and other relevant data.							
NOTE 2 Project specific companion specifications may specify a more precise meaning of the instances of the different event logs, i.e. the data captured and the number of events captured.							

Table B.4. 5: Event log object table [2].

The process of reading the mentioned attributes of all captured COSEM objects are shown in detail in Appendix C.

Additionally, in order to control the valve of the meter, the following steps must be taken (Refers to Section 5.4 and Appendix C). First, the "Clock" COSEM object is obtained using the *GXDLMSClock* class, the OBIS codes and the other specifications of this object defined in the Blue Book and UNI/TS [2], [39]. Then the second attribute of the “Clock” COSEM object is read, which is *Time*.

```
GXDLMSClock IC_clock = new GXDLMSClock();
result = reader.Read(IC_clock, 2);
```

The OBIS Codes of the “Clock” COSEM object in “UNI/TS”:

Clock	8	0		0-0:1.0.0.255	M	G	P	u	b	l	/	M	G	A
logical_name	m		1	octet-string		G						G		
time	m		2	octet-string		G/S						G/S		
time_zone	m		3	long	+60	G/S						G/S		
status	m		4	unsigned		G						G		
daylight_savings_begin	m		5	octet-string	0xFF 0xFF 0x03 0xFE 0x07 0x02 0x00 0x00 0x00 0x00 0x78 0x80	G/S						G/S		
daylight_savings_end	m		6	octet-string	0xFF 0xFF 0x0A 0xFE 0x07 0x03 0x00 0x00 0x00 0x00 0x3C 0x00	G/S						G/S		
daylight_savings_deviation	m		7	integer	+60	G/S						G/S		
daylight_savings_enabled	m		8	boolean	TRUE	G/S						G/S		
clock_base	o		9	enum	1	G						G		
adjust_to_quarter	o		1	integer										
adjust_to_measuring_period	o		2	integer										
adjust_to_minute	o		3	integer										
adjust_to_preset_time	o		4	integer										
preset_adjusting_time	o		5	integer										
shift_time	m		6	long		A								

Table B.4. 6: Clock object table [39].

Appendices

The "Single Action Schedule" COSEM object is obtained using the *GXDLMActionSchedule* class, the OBIS codes defined in the "Lists" class of the Visual Studio program and the other specifications of this object defined in the Blue Book and UNI/TS [2], [39]. Then the second attribute of the "Single Action Schedule" COSEM object is written (set), which is *Executed_Script*. This attribute has two data, "ExecutedScriptLogicalName" and "ExecutedScriptSelector".

The "ExecutedScriptSelector" can be set to "01" to open the valve and to "02" to close the valve.

```
GXDLMActionSchedule IC_actionschedule = new GXDLMActionSchedule(Lists.
DLMSParameters.Single_Action_Schedule);

IC_actionschedule.ExecutedScriptLogicalName = " 0.0.10.0.106.255";
if (change.value[0] == 0x01) { IC_actionschedule.ExecutedScriptSelector = 01; }
if (change.value[0] == 0x02) { IC_actionschedule.ExecutedScriptSelector = 02; }
reader.Write(IC_actionschedule, 2);
```

Then the fourth attribute of the "Single Action Schedule" COSEM object is also written (set), namely *Execution_Time*, which contains two data, namely "time" and "date" as codes below and Appendix C. The *Execution_Time* attribute indicates the time and date at which the script is executed.

```
//DateTime dt = new DateTime(IC_clock.Time.Value.DateTime.Year,
IC_clock.Time.Value.DateTime.Month, (IC_clock.Time.Value.DateTime.Day-1));
DateTime dt = IC_clock.Time.Value.DateTime;
dt = dt.AddDays(-1);
dt = dt.AddMinutes(-5);
GXDateTime[] dt_ = new GXDateTime[1];
dt_[0] = dt;
IC_actionschedule.ExecutionTime = dt_;
reader.Write(IC_actionschedule, 4);
change.IsAction = true;
_context.SaveChanges();
```

"Single Action Schedule" OBIS Codes in "List" class, "Blue Book", and "UNI/TS":

```
public const string Single_Action_Schedule = "0.0.15.0.1.255";
```

Single action schedule		0...n	class_id = 22, version = 0			
Attributes		Data type	Min.	Max.	Def.	Short name
1. logical_name	(static)	octet-string				x
2. executed_script	(static)	script				x + 0x08
3. type	(static)	enum				x + 0x10
4. execution_time	(static)	array				x + 0x18
Specific methods		m/o				

Table B.4. 7: Single Action Schedule object table [2].

Disconnect control Single action schedule	22	0			0-0:15.0.1.255	MG	Pub	I/M	GA
logical_name	m		1	octet-string		G	—	G	—
executed_script	m		2	script	0-0:10.0.106.255, (1,2,3)	G/S	—	G/S	—
type	o		3	enum	4	G	—	G	—
execution_time	m		4	array of 1	date, at "NOT SPECIFIED" time=0,0..	G/S	—	G/S	—

Table B.4. 8: Single Action Schedule object table [39].

Finally, the "Disconnect Control" COSEM object is obtained using the *GXDLMSDisconnectControl* class, the OBIS codes defined in the "Lists" class of the Visual Studio program and the other specifications of this object defined in the Blue Book and UNI/TS [2], [39]. Then the second attribute of the "Disconnect Control" COSEM object is read, which is *Output_State*. This attribute has two states, False (Disconnected valve) and True (Connected valve). After that, the results must be converted into a "byte array" and then stored in the "DLMS_Data_Record" table as the following codes.

```
GXDLMSDisconnectControl IC_disc = new GXDLMSDisconnectControl(Lists.
DLMSParameters.disc_control);
result = reader.Read(IC_disc, 2);
byte[] data = new byte[1];
data[0] = 0x10;

if ((bool)result == true) { data[0] = 0x01; }
if ((bool)result == false) { data[0] = 0x00; }
DB_write_records(strresult, 70, strLNtoarray(IC_disc.LogicalName), data, 0x00);
```

"Disconnect Control" OBIS Codes in "List" class and "Blue Book":

```
public const string disc_control = "0.0.96.3.10.255";
```

Disconnect control		0...n	class_id = 70, version = 0			
Attributes		Data type	Min.	Max.	Def.	Short name
1.	logical_name (static)	octet-string				x
2.	output_state (dyn.)	boolean				x + 0x08
3.	control_state (dyn.)	enum				x + 0x10
4.	control_mode (static)	enum				x + 0x18
Specific methods		m/o				
1.	remote_disconnect (data)	m				x + 0x20
2.	remote_reconnect (data)	m				x + 0x28

Table B.4. 9: Disconnect Control object table [2].

B.5 Storing the data in database

Regarding Appendix B.4, each object is stored in the database by using the “DB_write_records” method. The figure below depicts the storing data in the “DLMS_Data_Records” table based on the “DLMS_Connection_Sessions” table in the database, which was described in Appendix B.3.

The following codes are defined in the “DB_write_records” method:

```
public void DB_write_records(string LDN, int interface_class, byte[]  
logical_name, byte[] data, byte unit)
```

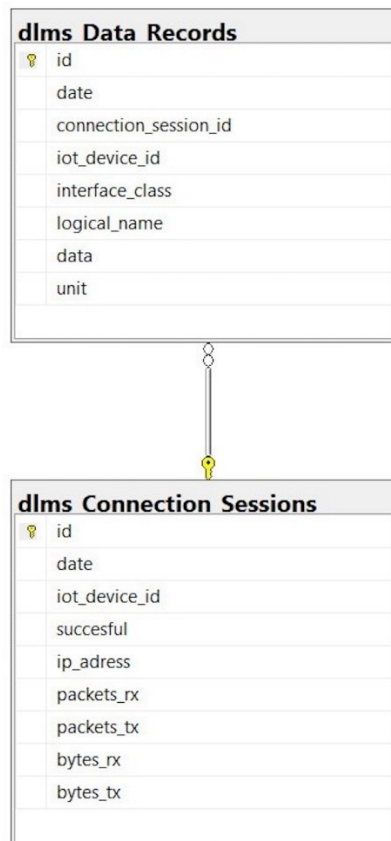


Figure B.5.1: DLMS_Connection_Sessions and DLMS_Data_Records diagrams in the database.

After recording all the objects in the database, the “DLMS_Connection_Sessions” table is edited and updated, and the session is written with true successful as codes below:

```
DB_edit_session(ip, true, (int)packetrx, (int)packettx, (int)media.  
BytesReceived, (int)media.BytesSent);
```

The following are defined in the “DB_edit_session” method:

Appendices

```
public void DB_edit_session(string ip, bool succesful, int packet_rx, int packet_tx, int byte_rx, int byte_tx)
```

A path is defined to save the log file after finishing the measurements and readings as follows:

```
int rand_name = rnd.Next(); // Create random number

string path = System.Environment.GetFolderPath
(Environment.SpecialFolder.Desktop); // location of the txt file(log)
string fileName = @path + "\\ " + rand_name + ".txt";

StreamWriter sw = System.IO.File.CreateText(fileName);
```

B.6 Close connection and disconnect DLMS

```
reader.Close(); // disconnect meter
media.Close(); // close tcp connection
```

Appendix C: DLMS messages translations

To continue with Section 6.1 (Log file translation), DLMS messages exchanged between the meter (server) and the software (client) in *Hex String* format are translated into *XML* format by using the "Gurux DLMS translator" as follows:

Initially, the client sends an *AssociationRequest (AARQ)* to the server. This request message is also illustrated in the third line of its translation in Figure C.1 as a DLMS message with *Hex String* format. All DLMS messages with *Hex String* format are shown in the third lines of their translation figures as the following figures and they are also saved with the exact time and date in the log file as two lines below.

```
3/21/2022 10:10:06 AM : TX:    10:10:06 AM    00 01 00 10 00 01 00 1F 60 1D A1 09 06 07 60
85 74 05 08 01 01 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 20 1E 5D FF FF
```

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 10 00 01 00 1F 60 1D A1 09 06 07 60 85 74 05 08 01 01 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 20 1E 5D FF FF
<WRAPPER len="1F" >
<TargetAddress Value="10" />
<SourceAddress Value="1" />
<PDU>
<AssociationRequest>
  <ApplicationContextName Value="LN" />
  <InitiateRequest>
    <ProposedDlmsVersionNumber Value="06" />
    <ProposedConformance>
      <ConformanceBit Name="GeneralBlockTransfer" />
      <ConformanceBit Name="BlockTransferWithGetOrRead" />
      <ConformanceBit Name="BlockTransferWithSetOrWrite" />
      <ConformanceBit Name="BlockTransferWithAction" />
      <ConformanceBit Name="MultipleReferences" />
      <ConformanceBit Name="Access" />
      <ConformanceBit Name="Get" />
      <ConformanceBit Name="Set" />
      <ConformanceBit Name="SelectiveAccess" />
      <ConformanceBit Name="Action" />
    </ProposedConformance>
    <ProposedMaxPduSize Value="FFFF" />
  </InitiateRequest>
</AssociationRequest>
</PDU>
</WRAPPER>
```

Figure C. 1: AssociationRequest (AARQ).

Then the server sends an *AssociationResponse (AARE)* to the client.

```
3/21/2022 10:10:08 AM : RX:    10:10:08 AM    00 01 00 01 00 10 00 2B 61 29 A1 09 06 07 60 85
74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 00 02 14 04
32 00 07
```



```

BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 10 00 2B 61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 00
02 14 04 32 00 07
<WRAPPER len="2B" >
<TargetAddress Value="1" />
<SourceAddress Value="10" />
<PDU>
<AssociationResponse>
<ApplicationContextName Value="LN" />
<AssociationResult Value="00" />
<ResultSourceDiagnostic>
<ACSEServiceUser Value="00" />
</ResultSourceDiagnostic>
<InitiateResponse>
<NegotiatedDlmsVersionNumber Value="06" />
<NegotiatedConformance>
<ConformanceBit Name="MultipleReferences" />
<ConformanceBit Name="Get" />
<ConformanceBit Name="SelectiveAccess" />
</NegotiatedConformance>
<NegotiatedMaxPduSize Value="0432" />
<VaaName Value="0007" />
</InitiateResponse>
</AssociationResponse>
</PDU>
</WRAPPER>

```

Figure C. 2: AssociationResponse (AARE).

Regarding the Section 5.2, the DLMS connection is established here without security (No Security level) for reading the "Logical Device Name". Thus, the client sends a *GetRequest* with the *Class_id=0001* (Data), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the "**Logical Device Name**" as the first COSEM object and read its second attribute (Value).

```

BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 10 00 01 00 0D C0 01 C1 00 01 00 00 2A 00 00 FF 02 00
<WRAPPER len="D" >
<TargetAddress Value="10" />
<SourceAddress Value="1" />
<PDU>
<GetRequest>
<GetRequestNormal>
<!-- Priority: High, ServiceClass: Confirmed, Invoke ID: 1 -->
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
<!-- Data -->
<ClassId Value="0001" />
<!-- 0.0.42.0.0.255 -->
<InstanceId Value="00002A0000FF" />
<!-- Value -->
<AttributeId Value="02" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
</PDU>
</WRAPPER>

```

Figure C. 3: GetRequest for LDN.

Then *GetResponse*, which contains the value of the "Logical Device Name" with the *string* data type, is sent from the server to the client. As the following figure, the "Logical Device Name" is "SMQU034A00000004".

Appendices

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 10 00 16 C4 01 C1 00 0A 10 53 4D 51 55 30 33 34 41 30 30 30 30 30 30 34
<WRAPPER len="16" >
<TargetAddress Value="1" />
<SourceAddress Value="10" />
<PDU>
<GetResponse>
<GetResponseNormal>
<!-- Priority: High, ServiceClass: Confirmed, Invoke ID: 1 -->
<InvokeIdAndPriority Value="C1" />
<Result>
<Data>
<String Value="SMQU034A00000004" />
</Data>
</Result>
</GetResponseNormal>
</GetResponse>
</PDU>
</WRAPPER>
```




Figure C. 4: *GetResponse* for LDN.

The client then sends *GetRequest* with the *Class_id=0001* (Data), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the "Metering Point ID" COSEM object and read its second attribute (Value).

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 10 00 01 00 0D C0 01 C1 00 01 00 00 60 01 0A FF 02 00
<WRAPPER len="D" >
<TargetAddress Value="10" />
<SourceAddress Value="1" />
<PDU>
<GetRequest>
<GetRequestNormal>
<!-- Priority: High, ServiceClass: Confirmed, Invoke ID: 1 -->
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
<!-- Data -->
<ClassId Value="0001" />
<!-- 0.0.96.1.10.255 -->
<InstanceId Value="000060010AFF" />
<!-- Value -->
<AttributeId Value="02" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
</PDU>
</WRAPPER>
```




Figure C. 5: *GetRequest* for "Metering Point ID".

The server responds to the client by sending *GetResponse*, which contains the value of the "Metering Point ID" with the *string* data type. As shown in the figure below, the "Metering Point ID" is "00000000".

```

BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 10 00 0E C4 01 C1 00 0A 08 30 30 30 30 30 30 30
<WRAPPER len="E" >
<TargetAddress Value="1" />
<SourceAddress Value="10" />
<PDU>
<GetResponse>
<GetResponseNormal>
<!-- Priority: High, ServiceClass: Confirmed, Invoke ID: 1 -->
<InvokeIdAndPriority Value="C1" />
<Result>
<Data>
<String Value="00000000" />
</Data>
</Result>
</GetResponseNormal>
</GetResponse>
</PDU>
</WRAPPER>

```

Figure C. 6: GetResponse for "Metering Point ID".

Regarding the Appendix B.2, The client then sends *GetRequest* with the *Class_id=0001* (Data), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the "Invocation Counter" COSEM object in "No Security" level and read its second attribute (Value).

```

BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 10 00 01 00 0D C0 01 C1 00 01 00 00 2B 01 01 FF 02 00
<WRAPPER len="D" >
<TargetAddress Value="10" />
<SourceAddress Value="1" />
<PDU>
<GetRequest>
<GetRequestNormal>
<!-- Priority: High, ServiceClass: Confirmed, Invoke ID: 1 -->
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
<!-- Data -->
<ClassId Value="0001" />
<!-- 0.0.43.1.1.255 -->
<InstanceId Value="00002B0101FF" />
<!-- Value -->
<AttributeId Value="02" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
</PDU>
</WRAPPER>

```

OBIS code of "Invocation Counter"

Figure C. 7: GetRequest for "Invocation Counter".

The server responds to the client by sending *GetResponse*, which contains the value of the "Invocation Counter" with the *Uint32* data type. As shown in the figure below, the "Invocation Counter" is "00000000".

Appendices

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 10 00 09 C4 01 C1 00 06 00 00 00 00
<WRAPPER len="9" >
  <TargetAddress Value="1" />
  <SourceAddress Value="10" />
  <PDU>
    <GetResponse>
      <GetResponseNormal>
        <!-- Priority: High, ServiceClass: Confirmed, Invoke ID: 1 -->
        <InvokeIdAndPriority Value="C1" />
        <Result>
          <Data>
            <UInt32 Value="00000000" />
          </Data>
        </Result>
      </GetResponseNormal>
    </GetResponse>
  </PDU>
</WRAPPER>
```

Figure C. 8: GetResponse for "Invocation Counter".

Then the client sends a *ReleaseRequest* to the server to disconnect.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 10 00 01 00 05 62 03 80 01 00
<WRAPPER len="5" >
  <TargetAddress Value="10" />
  <SourceAddress Value="1" />
  <PDU>
    <ReleaseRequest>
      <Reason Value="Normal" />
    </ReleaseRequest>
  </PDU>
</WRAPPER>
```

Figure C. 9: ReleaseRequest for disconnect.

The server replies with a *ReleaseResponse*. As a result, according to Section 5.2 and Appendix B.2, the connection is closed and a wait time of approximately two seconds takes place.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 10 00 17 63 11 80 01 00 BE 0F 04 0E 08 00 06 5F 1F 04 00 00 02 14 04 32 00 07
<WRAPPER len="17" >
  <TargetAddress Value="1" />
  <SourceAddress Value="10" />
  <PDU>
    <ReleaseResponse>
      <Reason Value="Normal" />
      <InitiateResponse>
        <NegotiatedDlmsVersionNumber Value="06" />
        <ProposedConformance>
          <ConformanceBit Name="MultipleReferences" />
          <ConformanceBit Name="Get" />
          <ConformanceBit Name="SelectiveAccess" />
        </NegotiatedConformance>
        <NegotiatedMaxPduSize Value="0432" />
        <VaaName Value="0007" />
      </InitiateResponse>
    </ReleaseResponse>
  </PDU>
</WRAPPER>
```

Figure C. 10: ReleaseResponse for disconnect.

Appendices

In accordance with Section 5.3 and Appendix B.3, the client sends the *AssociationRequest* (AARQ) to the server to establish a connection with the LN_With_Ciphering application context (High-Level-Security GMAC authentication mechanism or HighGMAC), and the client's "System title".

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 5F 60 5D A1 09 06 07 60 85 74 05 08 01 03 A6 0A 04 08 00 11 22 33 44 55 66 77 8A 02 07 80 8B 07 60 85 74 05 08 02
05 AC 12 80 10 0A 3F 1F 2F 4A 71 04 74 26 51 70 76 6A 63 01 4D BE 23 04 21 21 1F 30 00 00 00 00 19 6E C1 ED D3 2D 4A BE 94 55 5B 5A 3C
69 E0 4E 5F 2E 16 8A 9D 49 E2 60 5E 27
<WRAPPER len="E" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<AssociationRequest>
  <ApplicationContextName Value="LN_WITH_CIPHERING" />
  <CallingAP Title Value="0011223344556677" />
  <SenderACSERequirements Value="1" />
  <MechanismName Value="HighGMAC" />
  <CallingAuthentication Value="0A3F1F2F4A710474265170766A63014D" />
  <!-- Decrypted data -->
  Security: AuthenticationEncryption
  Invocation Counter: 0
  <InitiateRequest>
    <ProposedDlmsVersionNumber Value="06" />
    <ProposedConformance>
      <ConformanceBit Name="GeneralBlock Transfer" />
      <ConformanceBit Name="Block TransferWithGetOrRead" />
      <ConformanceBit Name="Block TransferWithSetOrWrite" />
      <ConformanceBit Name="Block TransferWithAction" />
      <ConformanceBit Name="MultipleReferences" />
      <ConformanceBit Name="Access" />
      <ConformanceBit Name="Get" />
      <ConformanceBit Name="Set" />
      <ConformanceBit Name="SelectiveAccess" />
      <ConformanceBit Name="Action" />
    </ProposedConformance>
    <ProposedMaxPduSize Value="0100" />
  </InitiateRequest>
  -->
  <glo_InitiateRequest Value="3000000000196EC1EDD32D4ABE94555B5A3C69E04E5F2E168A9D49E2605E27" />
</AssociationRequest>
</PDU>
</WRAPPER>
```

System title
for the software (client)

Figure C. 11: *AssociationRequest* (AARQ) for HighGMAC.

Therefore, from here on, all data packets are encrypted and decrypted by the *HighGMAC* mechanism based on the information such as the "Block Cipher Key", the "Authentication Key", the specified "AuthenticationEncryption security level", the "Invocation Counter", and the system titles of both the meter (server-side) and the software (client-side) (Refer to Sections 3.1.7.2.1, 4.4, and 5.3).

Then the server sends the *AssociationResponse* (AARE) to the client with the mentioned authentication mechanism and application context, and the server's "System title".

```

BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key: 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 6B 61 69 A1 09 06 07 60 85 74 05 08 01 03 A2 03 02 01 00 A3 05 A1 03 02 01 0E A4 0A 04 08 B1 4D 04 00 00 00 4A 03
88 02 07 80 89 07 60 85 74 05 08 02 05 AA 12 80 10 14 0A 85 C2 E1 F0 78 3C 1E 8F C7 63 31 18 8C 46 BE 23 04 21 28 1F 30 00 00 00 01 81 80
67 43 09 50 86 4D 6C 15 97 ED B9 CB 98 5C D3 E7 8D B9 51 32 1D 4F 68 71
<WRAPPER len="E" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<AssociationResponse>
<ApplicationContextName Value="LN_WITH_CIPHERING" />
<AssociationResult Value="00" />
<ResultSourceDiagnostic>
<!-- Authentication Required -->
<ACSEServiceUser Value="0E" />
</ResultSourceDiagnostic>
<!-- UNI/TS system title:
Manufacturer: SMQ
Serial number: 034A00000004
-->
<RespondingAPTitle Value="B14D040000004A03" />
<ResponderACSERequirement Value="1" />
<MechanismName Value="HighGMAC" />
<RespondingAuthentication Value="140A85C2E1F0783C1E8FC76331188C46" />
<!-- Decrypted data:
Security: AuthenticationEncryption
Invocation Counter: 1
InitiateResponse>
<NegotiatedDlmsVersionNumber Value="06" />
<NegotiatedConformance>
<ConformanceBit Name="MultipleReferences" />
<ConformanceBit Name="DataNotification" />
<ConformanceBit Name="Get" />
<ConformanceBit Name="Set" />
<ConformanceBit Name="SelectiveAccess" />
<ConformanceBit Name="Action" />
</NegotiatedConformance>
<NegotiatedMaxPduSize Value="0100" />
<VaaName Value="0007" />
</InitiateResponse>
-->
<glo_InitiateResponse Value="3000000001818067430950864D6C1597EDB9CB985CD3E78DB951321D4F6871" />
</AssociationResponse>
</PDU>
</WRAPPER>

```

**System title
for the meter (server)**

Figure C. 12: AssociationResponse (AARE) for HighGMAC.

Appendices

Then the client sends an *ActionRequest* to the server with the *class_id* value = 000F (Association LN), *Instance_id* (OBIS code), and *Method_id* = 01 (reply_to_HLS_authentication) in order to obtain the “**Association Logical Name (LN)**” COSEM object and read its first method which is the “Reply to the HLS authentication”.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 33 CB 31 30 00 00 00 01 96 67 89 57 02 F5 4E 82 EA 72 59 BE 63 B9 68 37 EA F2 53 90 44 8E 07 9D AA 6F CA D2 DA
DC 41 1A 4A 95 2F 55 74 76 6F D4 69 58 1B 0D
<WRAPPER len="33" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 1 -->
<!-- Decrypt data: C3 01 C1 00 0F 00 00 28 00 00 FF 01 01 09 11 10 00 00 00 01 21 B7 34 5A 93 2C BF 48 B3 0F 3E 66
<ActionRequest>
  <ActionRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeldAndPriority Value="C1" />
    <MethodDescriptor>
      # AssociationLogicalName
      <ClassId Value="000F" />
      # 0.0.40.0.0.255
      <InstanceId Value="0000280000FF" />
      # Reply to HLS authentication
      <MethodId Value="01" />
    </MethodDescriptor>
    <MethodInvocationParameters>
      <OctetString Value="100000000121B7345A932CBF48B30F3E66" />
    </MethodInvocationParameters>
  </ActionRequestNormal>
</ActionRequest>
-->
<glo_ActionRequest
Value="30000000019667895702F54E82EA7259BE63B96837EAF25390448E079DAA6FCAD2DA0C411A4A952F5574766FD469581B0D" />
</PDU>
</WRAPPER>
```

Figure C. 13: *ActionRequest* for “Association LN” and replying the “HLS authentication”.

The server responds to the client by sending the *ActionResponse*, which contains the value of the “Association Logical Name” with the *octet string* data type and the “Success” result of the connection establishment with “HLS authentication” as follows:

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 2C CF 2A 30 00 00 00 02 CC 42 C9 C1 36 39 01 8D 90 77 6D 39 9C DF DC 05 37 67 16 B9 A7 84 88 C4 44 4B A6 65 E2
63 DC E9 CC 64 44 51 05
<WRAPPER len="2C" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 2 -->
<!-- Decrypt data: C7 01 C1 00 01 00 09 11 10 00 00 00 02 C1 BB 27 B4 20 AD 19 81 66 29 BB 76
<ActionResponse>
  <ActionResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeldAndPriority Value="C1" />
    <Result Value="Success" />
    <ReturnParameters>
      <Data>
        <OctetString Value="1000000002C1BB27B420AD19816629BB76" />
      </Data>
    </ReturnParameters>
  </ActionResponseNormal>
</ActionResponse>
-->
<glo_ActionResponse Value="3000000002CC42C9C13639018D90776D399CDFDC05376716B9A78488C4444BA665E263DCE9CC64445105" />
</PDU>
</WRAPPER>
```

Figure C. 14: *ActionResponse* for “Association LN” and replying the “HLS authentication”.

Appendices

Then the client sends a *GetRequest* with the *Class_id=0001* (Data), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Logical Device Name**” COSEM object again and read its second attribute (Value), however this time the client requests this COSEM object with High-Level-Security authentication.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 02 A0 E3 B8 95 1E 87 4B B1 7B 6B D0 81 70 C8 02 7D B2 A6 12 1B FA 03 6A 3B FB
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 2 -->
<!-- Decrypt data: C0 01 C1 00 01 00 00 2A 00 00 FF 02 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Data
      <ClassId Value="0001" />
      # 0.0.42.0.0.255
      <InstanceId Value="00002A0000FF" />
      # Value
      <AttributeId Value="02" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
->
<glo_GetRequest Value="3000000002A0E3B8951E874BB17B6BD08170C8027DB2A6121BFA036A3BFB" />
</PDU>
</WRAPPER>
```

Figure C. 15: *GetRequest* for LDN with HighGMAC.

The server responds to the client by sending the *GetResponse*, which contains the value of the "Logical Device name" with the *string* data type.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 29 CC 27 30 00 00 00 03 A4 8E 0A 4F D9 70 76 E8 5B B2 41 28 0A 33 D8 10 EA CD 31 06 BC C2 A2 C8 25 14 F5 F5 81
03 2C 0E 1B C0
<WRAPPER len="29" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 3 -->
<!-- Decrypt data: C4 01 C1 00 0A 10 53 4D 51 55 30 33 34 41 30 30 30 30 30 34
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
      <String Value="SMQU034A00000004" />
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
->
<glo_GetResponse Value="3000000003A48E0A4FD97076E85BB241280A33D810EACD3106BCC2A2C82514F5F581032C0E1BC0" />
</PDU>
</WRAPPER>
```

Figure C. 16: *GetResponse* for LDN with HighGMAC.

Appendices

Then the other COSEM objects with “HighGMAC” authentication mechanism are obtained according to Section 5.4 and Appendix B.4.

The client sends a *GetRequest* with the *Class_id=0001* (Data), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Metering Point ID**” COSEM object again and read its second attribute (Value), however this time the client requests this COSEM object with High-Level-Security authentication.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 03 43 50 0F 25 C2 E1 D0 FA 5A 39 EC CD FE 32 80 AB 22 6C 14 5A 79 44 13 8E D7
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 3 -->
<!-- Decrypt data: C0 01 C1 00 01 00 00 60 01 0A FF 02 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Data
      <ClassId Value="0001" />
      # 0.0.96.1.10.255
      <InstanceId Value="000060010AFF" />
      # Value
      <AttributeId Value="02" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
-->
glo_GetRequest Value="300000000343500F25C2E1D0FA5A39ECCDFE3280AB226C145A7944138ED7" />
</PDU>
</WRAPPER>
```

Figure C. 17: *GetRequest* for "Metering Point ID" with HighGMAC.

The server responds to the client by sending the *GetResponse*, which contains the value of the "Metering Point ID" with the *string* data type.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 21 CC 1F 30 00 00 00 04 1C BA 6D 8F FB E7 74 10 B3 68 1C 9F 55 F5 03 95 B2 42 A9 93 33 A9 E9 0A DE 4E
<WRAPPER len="21" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 4 -->
<!-- Decrypt data: C4 01 C1 00 0A 08 30 30 30 30 30 30 30
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <String Value="00000000" />
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
-->
glo_GetResponse Value="30000000041CBA6D8FFBE77410B3681C9F55F50395B242A99333A9E90ADE4E" />
</PDU>
</WRAPPER>
```

Figure C. 18: *GetResponse* for "Metering Point ID" with HighGMAC.

Appendices

The client sends a *GetRequest* with the *Class_id=0003* (Register), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Absolute Converter Volume**” COSEM object and read its second attribute (Value).

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 04 D2 94 AB 79 33 BA 4F E2 3C 99 1D AF 75 E6 00 E2 A1 41 3D 5C 78 99 08 CD 5B
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 4 -->
<!-- Decrypt data: C0 01 C1 00 03 07 00 0D 02 00 FF 02 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Register
      <ClassId Value="0003" />
      # 7.0.13.2.0.255
      <InstanceId Value="07000D0200FF" />
      # Value
      <AttributeId Value="02" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
-->
<glo_GetRequest Value="3000000004D294AB7933BA4FE23C991DAF75E600E2A1413D5C789908CD5B" />
</PDU>
</WRAPPER>
```

Figure C. 19: *GetRequest* for "AbsoluteConverterVolume" with HighGMAC.

The server responds to the client by sending the *GetResponse*, which contains the value of the "Absolute Converter Volume" COSEM object with the *UInt32* data type.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 1C CC 1A 30 00 00 00 05 DA 84 D2 8A FD 3F A3 FE 40 FE A8 52 E0 E2 94 55 B1 FB 60 A7 CD
<WRAPPER len="1C" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 5 -->
<!-- Decrypt data: C4 01 C1 00 06 00 00 00 11
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <UInt32 Value="00000011" />
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
-->
<glo_GetResponse Value="3000000005DA84D28AFD3FA3FE40FEA852E0E29455B1FB60A7CD" />
</PDU>
</WRAPPER>
```

Figure C. 20: *GetResponse* for "AbsoluteConverterVolume" with HighGMAC.

Appendices

The client sends a *GetRequest* to the server in order to read also the third attribute of the “**Absolute Converter Volume**” COSEM object, which is the “Scaler and Unit” as mentioned in Appendix A and B.4.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 05 1D 32 45 68 BF A8 D3 A7 E0 72 62 E3 A2 51 A3 74 BE 38 8F E1 43 B0 19 F1 70
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 5 -->
<!-- Decrypt data: C0 01 C1 00 03 07 00 0D 02 00 FF 03 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Register
      <ClassId Value="0003" />
      # 7.0.13.2.0.255
      <InstanceId Value="07000D0200FF" />
      # Scaler and Unit
      <AttributeId Value="03" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
-->
<glo_GetRequest Value="30000000051D324568BFA8D3A7E07262E3A251A374BE388FE143B019F170" />
</PDU>
</WRAPPER>
```

Figure C. 21: *GetRequest* for the third attribute of "AbsoluteConverterVolume" with HighGMAC.

The server responds to the client by sending the *GetResponse* containing one *Structure* *Quantity=02*, which has two data with *Int8* and *Enum* data types. *Int8* is the data type of the “scaler” attribute and its value is “FF...D”, and *Enum* is the data type of the “unit” attribute and its value is “0E”, which shows the *corrected volume* (m³) [2].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 1D CC 1B 30 00 00 00 06 90 9B 27 D1 19 19 E5 28 79 B2 C8 32 17 B7 07 99 49 62 F4 AA DF 5F
<WRAPPER len="1D" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 6 -->
<!-- Decrypt data: C4 01 C1 00 02 02 0F FD 16 0E
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <Structure Qty="02" >
          <Int8 Value="FFFFFFFFFFFFFFFD" />
          <Enum Value="0E" />
        </Structure>
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
-->
<glo_GetResponse Value="3000000006909B27D11919E52879B2C83217B707994962F4AADF5F" />
</PDU>
</WRAPPER>
```

Figure C. 22: *GetResponse* for the third attribute of "AbsoluteConverterVolume" with HighGMAC.

Appendices

Then the client sends a *GetRequest* with the *Class_id=0003* (Register), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Current Diagnostic**” COSEM object and read its second attribute (Value).

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 06 86 6E 78 16 4F 49 69 CA 3D 36 8B 4F B7 3B 67 0B 40 C7 63 DC 6B FE F9 FC 4B
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 6 -->
<!-- Decrypt data: C0 01 C1 00 03 07 00 60 05 01 FF 02 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Register
      <ClassId Value="0003" />
      # 7.0.96.5.1.255
      <InstanceId Value="0700600501FF" />
      # Value
      <AttributeId Value="02" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
->
<glo_GetRequest Value="3000000006866E78164F4969CA3D368B4FB73B670B40C763DC6BFEF9FC4B" />
</PDU>
</WRAPPER>
```

Figure C. 23: *GetRequest* for "CurrentDiagnostic" with HighGMAC.

The server responds to the client by sending the *GetResponse* containing the value of the “Current Diagnostic” with the *UInt16* data type.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 1A CC 18 30 00 00 00 07 67 99 BE 71 F4 9D 1E 5F 00 A1 21 5A 8D E7 DA 4D 49 17 72
<WRAPPER len="1A" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 7 -->
<!-- Decrypt data: C4 01 C1 00 12 00 80
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <UInt16 Value="0080" />
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
->
<glo_GetResponse Value="30000000076799BE71F49D1E5F00A1215A8DE7DA4D491772" />
</PDU>
</WRAPPER>
```

Figure C. 24: *GetResponse* for "CurrentDiagnostic" with HighGMAC.

Appendices

The client sends a *GetRequest* with the *Class_id=0003* (Register), the *Attribute_id=02* (Value), and the *Instance_id* (OBIS code) to the server to obtain the “**Battery Estimated Remaining Use_0**” COSEM objects and read their second attribute (Value).

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 07 94 5F FC 73 88 25 9B AF 8B 32 D5 9E D4 BB 51 CB E4 A8 EA 2F 7C 97 02 2D 6A
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 7 -->
<!-- Decrypt data: C0 01 C1 00 03 00 00 60 06 06 FF 02 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Register
      <ClassId Value="0003" />
      # 0.0.96.6.6.255
      <InstanceId Value="0000600606FF" />
      # Value
      <AttributeId Value="02" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
->
<glo_GetRequest Value="3000000007945FFC7388259BAF8B32D59ED4BB51CBE4A8EA2F7C97022D6A" />
</PDU>
</WRAPPER>
```

Figure C. 25: *GetRequest* for "BatteryEstimatedRemainingUse_0" with HighGMAC.

The server responds to the client by sending the *GetResponse*, which contains the value of the “Battery Estimated Remaining Use_0” with the *UInt32* data type.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 1C CC 1A 30 00 00 00 08 B0 85 4C 57 CE AD 53 81 48 30 90 EC B8 A1 EE 1F 0C 12 6D 96 E3
<WRAPPER len="1C" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 8 -->
<!-- Decrypt data: C4 01 C1 00 06 1A BD 4E 70
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <UInt32 Value="1ABD4E70" />
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
->
<glo_GetResponse Value="3000000008B0854C57CEAD5381483090ECB8A1EE1F0C126D96E3" />
</PDU>
</WRAPPER>
```

Figure C. 26: *GetResponse* for "BatteryEstimatedRemainingUse_0" with HighGMAC.

Appendices

The client sends a *GetRequest* to the server in order to read also the third attribute of the “**Battery Estimated Remaining Use_0**” COSEM object, which is the “Scaler and Unit”.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 08 76 12 20 FA A2 01 12 4B 19 E0 BC 09 C5 76 CC E8 C8 98 49 E0 98 FF 8B F1 49
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 8 -->
<!-- Decrypt data: C0 01 C1 00 03 00 00 60 06 06 FF 03 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Register
      <ClassId Value="0003" />
      # 0.0.96.6.6.255
      <InstanceId Value="0000600606FF" />
      # Scaler and Unit
      <AttributeId Value="03" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
->
<glo_GetRequest Value="3000000008761220FAA201124B19E0BC09C576CCE8C89849E098FF8BF149" />
</PDU>
</WRAPPER>
```

Figure C. 27: *GetRequest* for the third attribute of "BatteryEstimatedRemainingUse_0" with HighGMAC.

The server responds to the client by sending the *GetResponse* containing one *Structure* *Quantity=02*, which has two data with *Int8* and *Enum* data types. *Int8* is the data type of the “scaler” attribute and its value is “00”, and *Enum* is the data type of the “unit” attribute and its value is “07”, which shows the *corrected volume* (m³) [2].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 1D CC 1B 30 00 00 00 09 FF 60 61 92 55 EF 16 3A 79 B5 12 F8 F9 9C 6F B5 C1 78 44 1A A4 C8
<WRAPPER len="1D" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 9 -->
<!-- Decrypt data: C4 01 C1 00 02 02 0F 00 16 07
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <Structure Qty="02" >
          <Int8 Value="00" />
          <Enum Value="07" />
        </Structure>
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
->
<glo_GetResponse Value="3000000009FF60619255EF163A79B512F8F99C6FB5C178441AA4C8" />
</PDU>
</WRAPPER>
```

Figure C. 28: *GetResponse* for the third attribute of "BatteryEstimatedRemainingUse_0" with HighGMAC.

Appendices

Similarly, for the second and third attributes of the "**Battery Estimated Remaining Use_1**" COSEM object, the same process is repeated.

The client sends a *GetRequest* with the *Class_id=0007* (Profile generic), the *Attribute_id=02* (*Buffer*), and the *Instance_id* (OBIS code) to the server to obtain the "**Metrological Event Logbook**" COSEM object and read its second attribute (*Buffer*) [2].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 0B 59 43 1C F5 ED 1A F0 9E E3 98 01 06 9A 75 32 06 7C CD AA CD 16 88 A7 BC 36
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 11 -->
<!-- Decrypt data: C0 01 C1 00 07 00 63 62 01 FF 02 00
<GetRequest>
<GetRequestNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
# ProfileGeneric
<ClassId Value="0007" />
# 7.0.99.98.1.255
<InstanceId Value="0700636201FF" />
# Buffer
<AttributeId Value="02" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
-->
<glo_GetRequest Value="300000000B59431CF5ED1AF09EE39801069A7532067CCDAACD1688A7BC36" />
</PDU>
</WRAPPER>
```

Figure C. 29: *GetRequest* for " MetrologicalEventLogbook " with HighGMAC.

The server sends a *GetResponse* to the client with the "Array Qty = 00", which is the data type of the "Buffer" attribute.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 19 CC 17 30 00 00 00 0C 05 B5 95 BA 7F FB 35 91 FE 7D 8E EC 53 A3 A0 6A D9 D5
<WRAPPER len="19" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 12 -->
<!-- Decrypt data: C4 01 C1 00 01 00
<GetResponse>
<GetResponseNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<Result>
<Data>
<Array Qty="00" >
</Array>
</Data>
</Result>
</GetResponseNormal>
</GetResponse>
-->
<glo_GetResponse Value="300000000C05B595BA7FFB3591FE7D8EEC53A3A06AD9D5" />
</PDU>
</WRAPPER>
```

Figure C. 30: *GetResponse* for " MetrologicalEventLogbook " with HighGMAC.

Appendices

The client sends a *GetRequest* to the server in order to read also the third attribute of the “**Metrological Event Logbook**” COSEM object, which is the “Capture_Objects” [2].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 0C 6B E0 69 C5 26 F2 B1 C2 74 FB AB 39 74 D7 3B 1C 91 26 CA 7C 38 F3 55 4F AF
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 12 -->
<!-- Decrypt data: C0 01 C1 00 07 07 00 63 62 01 FF 03 00
<GetRequest>
<GetRequestNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeldAndPriority Value="C1" />
<AttributeDescriptor>
# ProfileGeneric
<ClassId Value="0007" />
# 7.0.99.98.1.255
<InstanceId Value="0700636201FF" />
# CaptureObjects
<AttributeId Value="03" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
-->
<glo_GetRequest Value="300000000C6BE069C526F2B1C274FBAB3974D73B1C9126CA7C38F3554FAF" />
</PDU>
</WRAPPER>
```

Figure C. 31: *GetRequest* for the third attribute of " MetrologicalEventLogbook " with HighGMAC.

The server sends a *GetResponse* to the client with the “Array Qty = 05”, which is the data type of the “Capture_Objects” attribute and contains five data with *structure* data type, and each *structure* contains four data with *Octet String*, *Int8*, and two *Int16* data types as the figure below.

```
<!-- Invocation Counter: 13 -->
<!-- Decrypt data: C4 01 C1 00 01 05 02 04 12 00 01 09 06 00 00 01 01 00 FF 0F 02 1:
00 00 02 04 12 00 03 09 06 07 00 0D 02 00 FF 0F 02 12 00 00
<GetResponse>
<GetResponseNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeldAndPriority Value="C1" />
<Result>
<Data>
<Array Qty="05" >
<Structure Qty="04" >
<UInt16 Value="0001" />
# 0.0.1.1.0.255
<OctetString Value="0000010100FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0001" />
# 0.0.96.15.1.255
<OctetString Value="0000600F01FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0001" />
# 0.0.96.11.1.255
<OctetString Value="0000600B01FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0001" />
# 7.1.96.5.1.255
<OctetString Value="0701600501FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0003" />
# 7.0.13.2.0.255
<OctetString Value="07000D0200FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
</Array>
</Data>
</Result>
</GetResponseNormal>
</GetResponse>
```

Figure C. 32: *GetResponse* for the third attribute of " MetrologicalEventLogbook " with HighGMAC.

Appendices

After that, the “**Non-Metrological Event Logbook**” COSEM object is obtained, and all of its data are captured by reading its second attribute (Buffer). Then the client sends a *GetRequest* with the *Class_id=0007* (Profile generic), the *Attribute_id=03* (capture_objects), and the *Instance_id* (OBIS code) to the server in order to read the third attribute of the “**Non Metrological Event Logbook**” COSEM object, which is the “Capture_Objects” [2].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 0D A6 2A B9 22 DB 33 D3 73 43 CB 80 B6 FB 34 FB 7F 95 E2 91 86 FD DF 0B 81 52
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 13 -->
<!-- Decrypt data: C0 01 C1 00 07 07 00 63 62 00 FF 03 00
<GetRequest>
<GetRequestNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
# ProfileGeneric
<ClassId Value="0007" />
# 7.0.99.98.0.255
<InstanceId Value="0700636200FF" />
# CaptureObjects
<AttributeId Value="03" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
-->
<glo_GetRequest Value="300000000DA62AB922DB33D37343CB80B6FB34FB7F95E29186FDDF0B8152" />
</PDU>
</WRAPPER>
```

Figure C. 33: *GetRequest* for the third attribute of " NonMetrologicalEventLogbook " with HighGMAC.

The server sends a *GetResponse* to the client with the “Array Qty = 04”, which is the data type of the “Capture_Objects” attribute and contains four data with *structure* data type, and each *structure* contains four data with *Octet String*, *Int8*, and two *Int16* data types as the figure below.

```
<!-- Invocation Counter: 14 -->
<!-- Decrypt data: C4 01 C1 00 01 04 02 04 12 00 01 09 06 00 00 01 01 00 FF 0F 02 12 00 0
00 00
<GetResponse>
<GetResponseNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<Result>
<Data>
<Array Qty="04" >
<Structure Qty="04" >
<UInt16 Value="0001" />
# 0.0.1.1.0.255
<OctetString Value="0000010100FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0001" />
# 0.0.96.15.2.255
<OctetString Value="0000600F02FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0001" />
# 0.0.96.11.2.255
<OctetString Value="0000600B02FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
<Structure Qty="04" >
<UInt16 Value="0001" />
# 7.1.96.5.1.255
<OctetString Value="0701600501FF" />
<Int8 Value="02" />
<UInt16 Value="0000" />
</Structure>
</Array>
</Data>
</Result>
</GetResponseNormal>
</GetResponse>
```

Figure C. 34: *GetResponse* for the third attribute of " NonMetrologicalEventLogbook " with HighGMAC.

Appendices

For valve control processes, first, the client sends a *GetRequest* with the *Class_id=0008* (Clock), the *Attribute_id=02* (Time), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Clock**” COSEM object and read its second attribute, which is the “Time” [2].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 0D A6 2A B9 22 D4 34 D3 11 21 CB 80 B7 FB 00 47 24 43 7F 34 D7 C4 9F 9B D1 EF
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 13 -->
<!-- Decrypt data: C0 01 C1 00 08 00 01 00 00 FF 02 00
<GetRequest>
  <GetRequestNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      # Clock
      <ClassId Value="0008" />
      # 0.0.1.0.0.255
      <InstanceId Value="0000010000FF" />
      # Time
      <AttributeId Value="02" />
    </AttributeDescriptor>
  </GetRequestNormal>
</GetRequest>
->
<glo_GetRequest Value="300000000DA62AB922D434D31121CB80B7FB004724437F34D7C49F9BD1EF" />
</PDU>
</WRAPPER>
```

Figure C. 35: *GetRequest* for the second attribute of "Clock" with HighGMAC.

Then the server responds to the client by sending the *GetResponse*, which contains the value of the “Time” attribute with the *Octet String* data type. This value shows the current date and time of the meter.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 25 CC 23 30 00 00 00 0E 23 01 94 6A D3 40 38 74 B6 FD 4B DE 30 9E D5 54 F7 C6 2D 5D DA 9A 22 68 D0 93 A6 FA 4C 5C
<WRAPPER len="25" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 14 -->
<!-- Decrypt data: C4 01 C1 00 09 0C 07 E6 05 04 03 0B 25 29 00 00 3C 00
<GetResponse>
  <GetResponseNormal>
    # Priority: High, ServiceClass: Confirmed, Invoke ID: 1
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        # 5/4/2022 11:37:41 AM-01:00
        <OctetString Value="07E60504030B252900003C00" />
      </Data>
    </Result>
  </GetResponseNormal>
</GetResponse>
->
<glo_GetResponse Value="300000000E2301946AD3403874B6FD4BDE309ED554F7C62D5DDA9A2268D093A6FA4C5C" />
</PDU>
</WRAPPER>
```

Figure C. 36: *GetResponse* for the second attribute of "Clock" with HighGMAC.

Then the client sends a *SetRequest* with the *Class_id=0016* (Single Action Schedule), the *Attribute_id=02* (Executed_Script), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Single Action Schedule**” COSEM object and write (set) its second attribute, which

Appendices

is the "Executed_Script" with *script* data type [2], [39]. This attribute Also contains a structure, which has two data. The first data of the structure is "script_logical_name" with *Octet String* data type and the second data of the structure is "script_selector" with *UInt16* data type. Since the value of *UInt16* is "02", this means that the second state of the valve control is selected, namely the "open" state. Therefore, the meter valve is opened.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 2D C9 2B 30 00 00 0E 86 44 B2 57 CC FD A1 C7 5F 68 7B 0C 7E F8 23 2B 71 0D D5 77 3F 5A 87 7F 3F 3D 70 A3 7F 14 FE 66
46 11 A7 CB E4 B5
<WRAPPER len="2D" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 14 -->
<!-- Decrypt data: C1 01 C1 00 16 00 00 0F 00 01 FF 02 00 02 02 09 06 00 00 0A 00 6A FF 12 00 02
<SetRequest>
<SetRequestNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
# Action Schedule
<ClassId Value="0016" />
# 0.0.15.0.1.255
<InstanceId Value="00000F0001FF" />
# Executed Script Logical Name
<AttributeId Value="02" />
</AttributeDescriptor>
<Value>
<Structure Qty="02" >
# 0.0.10.0.106.255
<OctetString Value="00000A006AFF" />
<UInt16 Value="0002" />
</Structure>
</Value>
</SetRequestNormal>
</SetRequest>
-->
<glo_SetRequest Value="300000000E8644B257CCFDA1C75F687B0C7EF8232B710DD5773F5A877F3F3D70A37F14FE664611A7CBE4B5" />
</PDU>
</WRAPPER>
```

Figure C. 37: SetRequest for the second attribute of "Single Action Schedule" with HighGMAC.

The server sends a *SetResponse* to the client with the *Success* result value.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 17 CD 15 30 00 00 00 0F 70 9C FC C1 28 FB C1 9C 49 A3 73 59 EF E5 D7 2D
<WRAPPER len="17" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 15 -->
<!-- Decrypt data: C5 01 C1 00
<SetResponse>
<SetResponseNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<Result Value="Success" />
</SetResponseNormal>
</SetResponse>
-->
<glo_SetResponse Value="300000000F709CFCC128FBC19C49A37359EFE5D72D" />
</PDU>
</WRAPPER>
```

Figure C. 38: SetResponse for the second attribute of "Single Action Schedule" with HighGMAC.

Then the client sends a *SetRequest* with the *Class_id=0016* (Single Action Schedule), the *Attribute_id=04* (Executed_Time), and the *Instance_id* (OBIS code) to the server in order to write

Appendices

(set) the fourth attribute of the “**Single Action Schedule**” COSEM object, which is the “Executed_Time” [2], [39]. This attribute contains the *Array Qty=01*, which has one data, namely the *Structure Quantity=02*, which has two data, namely “time” and “date” with *Octet String* data types. Therefore, the “time” and the “date” are set as the following values.

```
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 31 C9 2F 30 00 00 00 0F BE B5 A8 9C E5 E7 26 D7 33 E6 D1 9F EC 9E A7 06 25 26 8A A9 60 11 2F 1C 69 A3 BD 47
D9 3A B8 05 9C FD A0 BC 9E AD 85 B2 E6 16
<WRAPPER len="31" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 15 -->
<!-- Decrypt data: C1 01 C1 00 16 00 00 0F 00 01 FF 04 00 01 01 02 02 09 04 0B 20 29 FF 09 05 07 E6 05 03 02
<SetRequest>
<SetRequestNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
# ActionSchedule
<ClassId Value="0016" />
# 0.0.15.0.1.255
<InstanceId Value="00000F0001FF" />
# Execution Time
<AttributeId Value="04" />
</AttributeDescriptor>
<Value>
<Array Qty="01" >
<Structure Qty="02" >
# 11:32:41 AM
<OctetString Value="0B2029FF" />
# 5/3/2022
<OctetString Value="07E6050302" />
</Structure>
</Array>
</Value>
</SetRequestNormal>
</SetRequest>
-->
<glo_SetRequest Value="300000000FBEB5A89CE5E726D733E6D19FEC9EA70625268AA960112F1C69A3BD47D93AB8059CFDA0BC9EAD85B2E616" />
</PDU>
```

Figure C. 39: SetRequest for the fourth attribute of "Single Action Schedule" with HighGMAC.

The server sends a *SetResponse* to the client with the *Success* result value

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 17 CD 15 30 00 00 00 10 95 43 51 3C A1 C1 3E FC 11 D4 CE 61 77 A6 5A 4B
<WRAPPER len="17" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 16 -->
<!-- Decrypt data: C5 01 C1 00
<SetResponse>
<SetResponseNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<Result Value="Success" />
</SetResponseNormal>
</SetResponse>
-->
<glo_SetResponse Value="30000000109543513CA1C13EFC11D4CE6177A65A4B" />
</PDU>
</WRAPPER>
```

Figure C. 40: SetResponse for the fourth attribute of "Single Action Schedule" with HighGMAC.

Appendices

Then the client sends a *GetRequest* with the *Class_id=0046* (Disconnect control), the *Attribute_id=02* (Output_State), and the *Instance_id* (OBIS code) to the server in order to obtain the “**Disconnect Control**” COSEM object and read its second attribute, which is the “Output_State” [2], [39].

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 20 C8 1E 30 00 00 00 10 4F 2E 1D CB 5D F0 5E 1F 83 2B 5D 4D 94 07 A1 7D 3B 35 32 61 05 A9 2A 2C E9
<WRAPPER len="20" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 16 -->
<!-- Decrypt data: C0 01 C1 00 46 00 00 60 03 0A FF 02 00
<GetRequest>
<GetRequestNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<AttributeDescriptor>
# DisconnectControl
<ClassId Value="0046" />
# 0.0.96.3.10.255
<InstanceId Value="000060030AFF" />
# Output State
<AttributeId Value="02" />
</AttributeDescriptor>
</GetRequestNormal>
</GetRequest>
->
<glo_GetRequest Value="30000000104F2E1DCB5DF05E1F832B5D4D9407A17D3B35326105A92A2CE9" />
</PDU>
</WRAPPER>
```

Figure C. 41: *GetRequest* for the second attribute of "Disconnect Control" with HighGMAC.

Then the server responds to the client by sending the *GetResponse*, which contains the value of the “Output State” attribute with the *Boolean* data type. This value is “false” and shows the “close” or “disconnect” state of the meter valve.

```
BlockCipher key: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00
Authentication Key:00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
1: 00 01 00 01 00 01 00 19 CC 17 30 00 00 00 11 FD 6B 8C 2A 54 27 34 E0 E5 EB BE 90 EE 77 30 BA C4 B9
<WRAPPER len="19" >
<TargetAddress Value="1" />
<SourceAddress Value="1" />
<PDU>
<!-- Invocation Counter: 17 -->
<!-- Decrypt data: C4 01 C1 00 03 00
<GetResponse>
<GetResponseNormal>
# Priority: High, ServiceClass: Confirmed, Invoke ID: 1
<InvokeIdAndPriority Value="C1" />
<Result>
<Data>
<Boolean Value="false" />
</Data>
</Result>
</GetResponseNormal>
</GetResponse>
->
<glo_GetResponse Value="3000000011FD6B8C2A542734E0E5EBBE90EE7730BAC4B9" />
</PDU>
</WRAPPER>
```

Figure C. 42: *GetResponse* for the second attribute of "Disconnect Control" with HighGMAC.

Appendix D: Abbreviations and acronyms

2G	Second Generation
AA	Application Association
AAD	Additional Authentication Data
AARE	A-Associate Response
AARQ	A-Associate Request
ACSE	Application Control Service Element
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AES-GCM	Advanced Encryption Standard-Galois/Counter Mode
AL	Application Layer
AMI	Advanced Metering Infrastructure
AMR	Automatic Meter Reading
ANSI	American National Standards Institute
AP	Application Process
APDU	Application Protocol Data Units
ASN.1	Abstract Syntax Notation One
ASP.NET	Active Server Pages Network Enabled Technologies
AT	ATtention
CF	Control Function
COSEM	Companion Specification for Energy Metering
CSV	Comma-Separated Values
CT	Clear Terminal
Class_id	Interface Class Identification Code
CtoS	The challenge of the Client to the Server during the HLS authentication
DLMS	Device Language Message Specification
DLMS UA	Device Language Message Specification User Association
DSL	Digital Subscriber Line
ECDSA	Elliptic Curve Digital Signature Algorithm is a cryptographic algorithm
ETSI	European Telecommunications Standard
FTP	File Transfer Protocol
GCM	Gallois Counter Mode
GMAC	Galois Message Authentication Code
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
GXNet class	Gurux.Net package from Gurux library

Appendices

HDLC	High-Level Data Link Control
HLS	High-Level-Security
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IC	Interface Class
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IIS	Internet Information Services
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
Int	Integer
IoT	Internet of Things
LCD	Liquid Crystal Display
LDN	Logical Device Name
LLS	Low-Level-Security
LN	Logical Name
LTE-M	LTE for Machine Type Communication
M2M	Machine to Machine
MCU	Microcontroller Unit
MD5	Message Digest Algorithm
MQTT	Message Queuing Telemetry Transport
NB-IoT	Narrowband IoT
OBIS	Object Identification System
OSGP	Open Smart Grid Protocol
OSI	Open Systems Interconnection
PLC	Power Line Communication
PPP	Point-to-point protocol
PSTN	Public Switched Telephone Network
Pub-Sub	Publish–subscribe pattern
RF Mesh	Radio Frequency Mesh
RLRE	A-Release Response
RLRQ	A-Release Request
RX	Receiver
SAP	Service Access Point

Appendices

SHA-1	Secure Hash Algorithm 1
SIM	Subscriber Identity Module
SM	Smart Meter
SN	Short Name
SQL	Structured Query Language
SaaS	Software as a Service
StoC	The challenge of the Server to the Client during the HLS authentication
TCP/IP	Transmission Control Protocol/ Internet Protocol
TX	Transmitter
UDP	User Datagram Protocol
UInt	Unsigned Integer
UNI/TS	Italian National Unification /Technical Specification
WiMAX	Worldwide Interoperability for Microwave Access
WPort	Wrapper Port
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
xDLMS ASE	Extended DLMS Application Service Element

References

- [1] A. Irfan et al., “IoT Based Smart Meter,” 3rd Int. Conf. Innov. Comput. ICIC 2019, no. Icic, 2019, doi: 10.1109/ICIC48496.2019.8966684.
- [2] “Blue Book-COSEM Interface Classes and OBIS Object Identification System.” <https://www.dlms.com/files/Blue-Book-Ed-122-Excerpt.pdf> (accessed Apr. 18, 2022).
- [3] M. Kozole, “Security in DLMS A White Paper by the DLMS User Association,” 2019.
- [4] I. O. Alwaan, “Smart Meter Reading Based on DLMS/COSEM Protocol,” 2017.
- [5] “Analysis of DLMS Protocol Technical Report.” <https://www.fit.vut.cz/research/publication-file/11616/TR-DLMS.pdf>.
- [6] “DLMS User Association.” <https://www.enlit-europe.com/exhibitors/dlms-user-association>.
- [7] “IEC_62056.” https://en.wikipedia.org/wiki/IEC_62056.
- [8] “TC 13-Electrical energy measurement and control.” https://www.iec.ch/dyn/www/f?p=103:7:716359149216685:::FSP_ORG_ID,FS P_LANG_ID:1258,25.
- [9] O. Hersent, D. Boswarthick, and E. Omar, The internet of things- Applications to the smart grid and building automation. .
- [10] “Green Book-DLMS/COSEM Architecture and Protocols.” <https://www.dlms.com/files/Green-Book-Ed-83-Excerpt.pdf> (accessed Apr. 18, 2022).
- [11] “Smart metering.” <https://www.quectel.com/library/why-cellular-connectivity-provides-the-robust-secure-foundation-for-new-revenues-in-smart-metering>.
- [12] “Industrial IoT Solutions.” <https://www.leewayhertz.com/industrial-iot-solutions/>.
- [13] “Smart meter.” https://en.wikipedia.org/wiki/Smart_meter.
- [14] “Automatic meter reading.” https://en.wikipedia.org/wiki/Automatic_meter_reading.

- [15] “Impact of IoT on smart metering.” <https://www.leewayhertz.com/iot-based-smart-metering/>.
- [16] “Smart metering solution.” <https://www.webnms.com/iot/smart-metering.html>.
- [17] “IoT Smart Metering Solutions.” https://www.kaaiot.com/iot-dashboards/smart-metering#for_solution_integrators.
- [18] “IoT Platform Characteristics.” <https://wolkabout.com/blog/five-most-important-iot-platform-characteristics/>.
- [19] “Communication technologies in smart metering.” <https://m2mserver.com/en/communications-technologies-in-smart-metering/>.
- [20] “An Introduction to Smart Meter Communication.” <https://www.emnify.com/blog/smart-meter>.
- [21] L. Štastný, L. Franek, and P. Fiedler, “Wireless communications in smart metering,” IFAC Proc. Vol., vol. 12, no. PART 1, pp. 330–335, 2013, doi: 10.3182/20130925-3-CZ-3023.00035.
- [22] “M2M Communication.” <https://www.rfwireless-world.com/>.
- [23] “LoRa Alliance.” <https://lora-alliance.org/>.
- [24] “Dlms overview.” <https://www.dlms.com/dlms-cosem/overview>.
- [25] C. Gomez, A. Arcia-moret, and J. Crowcroft, “TCP in the Internet of Things : from ostracism to prominence,” pp. 1–16.
- [26] “On-Premises vs. Cloud.” <https://www.morefield.com/blog/on-premises-vs-cloud/>.
- [27] “On-premises software.” https://en.wikipedia.org/wiki/On-premises_software.
- [28] “What is Cloud ERP Software?” <https://www.acumatica.com/what-is-cloud-erp-software/>.
- [29] “On Premise Licensing.” <https://cpl.thalesgroup.com/software-monetization/on-premise-licensing>.
- [30] “The Advantages and Disadvantages of Using Cloud-based Software Systems.” <https://www.bizprac.com/advantages-disadvantages-using-cloud-based-software-systems/>.

- [31] International Electrotechnical Commission, IEC 62056-6-1, 3.0. 2017.
- [32] International Electrotechnical Commission, IEC 62056-6-2, 3.0. 2017.
- [33] International Electrotechnical Commission, IEC 62056-5-3, 3.0. 2017.
- [34] P. Shanmukesh, L. Mahendra, K. JaganMohan, R. K. S. Kumar, and B. S. Bindhumadhava, "Secure DLMS/COSEM communication for Next Generation Advanced Metering Infrastructure," *ASIAN J. Converg. Technol.*, 2021, doi: 10.33130/ajct.2020v07i01.020.
- [35] N. Lüring, D. Szameitat, S. Hoffmann, and G. Bumiller, "Analysis of security features in DLMS/COSEM: Vulnerabilities and countermeasures," 2018, doi: 10.1109/ISGT.2018.8403340.
- [36] P. Shanmukesh, L. Mahendra, K. JaganMohan, R. K. S. Kumar, and B. S. Bindhumadhava, "Secure DLMS/COSEM communication for Next Generation Advanced Metering Infrastructure," *ASIAN J. Converg. Technol.*, vol. 7, no. 1, 2021, doi: 10.33130/ajct.2021v07i01.020.
- [37] S. H. Ju and H. S. Seo, "Design key management system for DLMS/COSEM standardbased smart metering," *Int. J. Eng. Technol.*, vol. 7, no. 3.34 Special Issue 34, 2018, doi: 10.14419/ijet.v7i3.34.19380.
- [38] "Gurux-DLMS Secure." <https://www.gurux.fi/Gurux.DLMS.Secure>.
- [39] UNI1603299, Italian standard- UNI/TS 11291-12-2 post CTC. 2018.
- [40] C. Protection and W. Paper, "Monitoring and Protecting Smart Meter Circuitry and Communications circuit protection white paper."
- [41] F. Gumyusenge, J. Mukamana, R. Mugisha, A. A. Garba, and M. Saint, *Design and Implementation of a Smart Meter*. Springer International Publishing, 2018.
- [42] T. Italmno, Italian standard- UNI/TS 11291-11-2 post CTC. 2014.