



Security Auditing of a DLMS/COSEM Smart Grid Communication Protocol Implementation

Henrique Califórnia Mendes

Mestrado em Engenharia Informática
Especialização em Arquitectura, Sistemas e Redes de Computadores

Dissertação orientada por:
Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves
Prof. Doutora Ibéria Vitória de Sousa Medeiros

Resumo

Em anos recentes, empresas de utilidade pública e governos de diversos países têm vindo a promover a renovação e melhoria das infraestruturas de energia, de modo a aumentar a eficiência e funcionalidade da rede. As primeiras iterações da rede eléctrica contavam apenas com contadores de electricidade locais que precisavam de ser lidos por um profissional no local. Os custos e falta de eficiência deste sistema foram as motivações principais para a evolução da tecnologia presente na rede. Com o avançar da tecnologia, tornou-se possível efectuar a leitura dos valores dos medidores remotamente, aumentando a eficiência da rede.

A rede eléctrica inteligente está prevista ser instalada num futuro próximo em muitos países existindo já países com regulações pertinentes que estipulam o tratamento, armazenamento, acesso e transporte dos dados. Esta rede será possível mediante a instalação de medidores inteligentes, capazes de comunicar com a rede e com outros dispositivos inteligentes presentes em habitações. Há contudo um receio justificado relativamente à introdução de problemas de segurança e privacidade, quer nos novos equipamentos que estarão fisicamente acessíveis em muitos casos, quer na rede em si, e da comunicação que nela irá ocorrer.

O protocolo DLMS/COSEM é um protocolo especificamente desenvolvido para a smart grid, e é, de entre outras opções, o que tem vindo a ganhar mais popularidade devido principalmente a poder ser usado para a medição de energia, gás natural, ou água. A sua especificação separa os meios e tecnologias de comunicação da modelação dos medidores, dos seus dados e sua representação, o que permite que estas sejam desenvolvidas independentemente, permitindo assim que o protocolo se adapte à evolução assíncrona destas duas partes. Isto permite que o suporte seja estendido para suportar mais meios tecnológicos e pilhas de protocolos de comunicação que surgirão no futuro. Um dos meios que está a ser bastante considerado pela indústria eléctrica é o de *power-line communication*, devido à possibilidade de usar os próprios meios de distribuição de energia para comunicar com os equipamentos da rede. Este protocolo goza também da disponibilidade de variados mecanismos de segurança, nomeadamente cifra autenticada de modo a permitir confidencialidade e autenticidade nas comunicações, assinatura digital para garantir a integridade de dados, autenticação por chave pública de forma a permitir aos nós

da rede definirem chaves criptográficas entre si, transporte de chaves criptográficas de forma segura, e compressão.

Devido à natureza de necessidade crítica que representa para a sociedade, e em semelhança com a rede atual, o seu correcto funcionamento deve ser assegurado com elevada confiabilidade. Para este fim, é necessário aferir se os protocolos usados para comunicação na rede são seguros na sua especificação, e posteriormente comprovar que a implementação do protocolo usada cumpre essa especificação. Se a rede eléctrica inteligente for implementada de forma insegura, pode dar azo a uma panóplia de catástrofes, desde ataques à privacidade e espionagem, até à paralização de serviços críticos como hospitais e prejuízo económico elevado.

A presente dissertação tem o intuito principal de avaliar a segurança do protocolo DLMS/COSEM, e de uma implementação do protocolo disponibilizada por um parceiro importante da indústria de energia eléctrica portuguesa.

Para levar a cabo o objetivo da tese, numa primeira fase, foi analisada a especificação do protocolo, com ênfase na segurança. Foram analisados com rigor especial as secções pertinentes aos mecanismos de encriptação autenticada e de autenticação utilizados na comunicação DLMS/COSEM. Também, foi realizado um levantamento das vulnerabilidades e fraquezas do protocolo já descritas na literatura existente e foi verificado que algumas delas ainda não tinham sido corrigidas ou colmatadas na versão mais recente da especificação do protocolo. Por exemplo, a especificação ainda permite o uso de algoritmos criptográficos atualmente considerados inseguros ou fracos, o que pode levar a ataques de *downgrade* mesmo havendo opções de algoritmos criptográficos melhores na especificação. Verificou-se também que existia pouco trabalho científico em torno do assunto, perante a iminente chegada desta tecnologia aos mercados de energia eléctrica de muitos países.

Para atingir esse objectivo foi desenhada a *framework* ValiDLMS, a qual é estratificada para a integração de ferramentas que permitem validar implementações quanto ao seu cumprimento da especificação do protocolo e testar a segurança das mesmas. Um protótipo da *framework* foi desenvolvido e avaliado usando tráfego DLMS/COSEM do referido parceiro.

Numa segunda fase, e para atingir o objetivo de testar a implementação do protocolo segundo uma especificação, foi desenhada a plataforma arquitectural *ValiDLMS*, a qual é estratificada para a integração de ferramentas que permitem validar implementações do protocolo segundo a especificação e testar a segurança destas. Tal arquitectura foi desenhada com base no conhecimento adquirido da literatura relevante existente, nos conhecimentos adquiridos na observação da implementação que nos foi disponibilizada, e do processo de desenvolvimento de uma ferramenta usando a nossa arquitectura. A arquitectura funciona escutando a rede e analisando o tráfego capturado à procura de anomalias, podendo também gerar tráfego de modo a causar a ocorrência de anomalias que

não seriam detectadas apenas por observação. Isto permite que ferramentas que sigam a arquitectura sejam usadas de modo passivo e activo, e também em ambientes reais para monitorização de uma rede de energia.

A arquitectura ValidDLMS é composta por três camadas, nomeadamente, ambiente DLMS/COSEM, interacção, e teste. Esta estratificação permite maior facilidade de integração de ferramentas, sendo que estas posicionam-se de acordo com as funcionalidades das camadas. A camada de ambiente DLMS/COSEM representa o ambiente onde se encontra a implementação DLMS/COSEM a ser testada pela ferramenta, ou seja, onde estão incluídos os meios físicos de comunicação, medidores de energia eléctrica inteligente, e concentradores de dados. A camada de interacção inclui dois componentes essenciais à interacção da ferramenta com o ambiente DLMS/COSEM, um componente *sniffer* que consegue capturar todo o tráfego DLMS/COSEM que ocorre no ambiente a ser testado, e um componente *DLMS client* capaz de estabelecer ligações e interagir completamente com os equipamentos presentes no ambiente, executando as ordens que recebe da camada de teste. A camada de teste usa uma base de dados para gerar interacções pseudo-aleatórias de modo a causar um comportamento errado do sistema e expor vulnerabilidades que não poderiam ser encontradas através da mera observação e adicionalmente investiga a fundo todo o tráfego capturado.

Tendo por base a arquitetura, foi contruída uma ferramenta que comporta um *add-on* para o programa de análise de tráfego *Wireshark*, um dissecador de tráfego do protocolo DLMS/COSEM, e um validador da implementação do protocolo programado, validando-o quer em especificação, quer em segurança. Em execução a ferramenta analisa os pacotes capturados pelo *Wireshark* e são criados registos dos problemas de segurança e dos problemas de não cumprimento da especificação do protocolo detectados, de forma visível no *Wireshark*, permitindo assim uma análise eficiente.

A ferramenta foi testada numa rede de teste do nosso parceiro da indústria de energia, constituída por dois concentradores de dados e seis medidores inteligentes. A ferramenta encontrou dois problemas nas mensagens DLMS/COSEM. Em mensagens de criação de ligações foi detectado um byte adicional entre campos da mensagem que não constava na especificação e que inicialmente dificultou a correcta interpretação da ferramenta. Adicionalmente, foi detectada uma palavra-chave de autenticação a ser transmitida em claro no tráfego capturado. A intercepção dessa chave por parte de um atacante permitiria a impersonificação do emissor, o que poderia ser usado para obter informações do uso de energia, ou mesmo enviar comandos para o equipamento da rede, dependendo das permissões do emissor original. Perante os resultados, a arquitectura desenvolvida apresenta o potencial desejado para ser utilizada na indústria, e para preencher o vácuo de investigação académica notado, quer em relação à segurança da rede eléctrica inteligente, quer em relação à segurança do protocolo DLMS/COSEM.

Palavras-chave: Protocolo DLMS/COSEM, Segurança da Rede Eléctrica Inteligente, Segurança e Privacidade, Vulnerabilidades, Infraestruturas Críticas

Abstract

The electrical grid is a critical infrastructure for modern society. It has been evolving into a smart(er) grid, allowing infrastructure aware decisions based on data collected in real-time from smart meters and other devices. This enables better monitoring and management solutions which would result in higher efficiency in matching generation and consumption. Smart meters and their uplinks have, however, limited physical security due to their location within customer premises. Many of the proposed smart grid communication protocols are also being evaluated because of other security and privacy concerns, which were raised mainly due to the high probability for misuse of the extended capabilities of smart meters.

DLMS/COSEM is one of the emerging communication protocols designed for use in the smart grid, mainly for its energy-type-agnostic interface model, and the separation from the communication media which allows future expandability. The protocol allows remote interactions with smart meters, often being deployed above power-line communication links.

The present dissertation aims to contribute to validity and security assessments of the DLMS/COSEM protocol and its implementations. In a first instance, an overview of the protocol is presented, followed by a compilation of information about the relevant vulnerabilities and attacks exposed in the existing literature. Then, we explain the design of our ValiDLMS framework, the first open source solution for validation and security auditing of DLMS/COSEM implementations using this communication profile. The framework was developed as an extension to *Wireshark* and was used to analyse an industry partner's DLMS/COSEM implementation. The results show that ValiDLMS can effectively support the discovery of bugs and/or other non-conformance problems.

Keywords: DLMS/COSEM protocol, Smart Grid Security, Security and Privacy, Vulnerabilities, Critical Infrastructures

Contents

List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Motivation	2
1.2 Objectives	3
1.3 Contributions	4
1.4 Organization	4
2 Context and Related Work	7
2.1 Advanced Metering Infrastructure	7
2.2 DLMS/COSEM Protocol	10
2.2.1 Overview	10
2.2.2 Security	12
2.2.3 Vulnerabilities	19
2.2.4 Attacks	24
3 The ValidDLMS Framework	27
3.1 Framework Overview	27
3.2 Interaction layer	28
3.2.1 DLMS Client	30
3.2.2 Sniffer	30
3.3 Testing layer	30
3.3.1 Fuzzing module	31
3.3.2 Verification Module	32
4 Proof of Concept	37
4.1 Testing Environment	37
4.2 ValidDLMS Implementation	37
4.3 Experimental Evaluation	38
4.3.1 Validation	39

4.3.2	Security auditing	41
4.3.3	Industry DLMS/COSEM implementations	41
5	Conclusion	43
	Bibliografia	47
	Índice	47

List of Figures

2.1	The SGAM smart grid plane [17].	8
2.2	AMI simplified topology [15, 29].	9
2.3	DLMS/COSEM simplified model and communication pattern.	12
2.4	Encoded DLMS APDU used to get the time attribute from a clock object.	12
2.5	Encoded DLMS APDU response for the one in Figure 2.4.	13
2.6	Overview of the levels of authentication supported by DLMS/COSEM.	13
2.7	Overview of the inputs and outputs of GCM [10, 12].	15
2.8	Overview of GCM authenticated encryption [12].	16
2.9	Structure of general-signing APDUs in DLMS/COSEM [10].	17
2.10	Overview of the elliptic curve key agreement schemes present in DLMS/COSEM [6, 10].	18
2.11	Overview of block structure of the authentication data used in GHASH to create authentication tags when also using encryption [29].	18
2.12	Overview of the algorithm used for protecting APDUs in DLMS/COSEM [10].	19
2.13	Procedure used to remove authentication from APDUs [29].	24
3.1	Overview of the ValiDLMS architecture for DLMS/COSEM testing tools.	29
3.2	Attack DB entry example.	32
3.3	DLMS packet with content structured as a tree, for an example get request packet validation.	33
4.1	Packets marked in wireshark using colour schemes.	40
4.2	Packet details with messages indicating problems found.	40
4.3	Log entries produced by the tool.	41

List of Tables

2.1	Simplified description of SGAM zones [17].	9
2.2	Available HLS authentication mechanisms [10].	14
2.3	Available security suites [10].	15

Chapter 1

Introduction

Electricity has become essential to human life in modern society. Thus the electrical grid is a critical infrastructure for society and without it many vital sectors would cease normal operation (e.g., communication, transportation, finances). Over the past century, utility grid technology has seen gradual advances, with the introduction of automatic meter reading (AMR), which allows utilities to collect readings that metering equipments send periodically, removing thus the need of having to send meter reading personnel to customer premises. More recently, these advances are being done at a higher pace in order to build what is called as the smart grid.

The European Committee for Electrotechnical Standardization (CENELEC) defines smart grid as "an electricity network that can cost efficiently integrate the behaviour and actions of all users connected to it – generators, consumers and those that do both – in order to ensure economically efficient, sustainable power system with low losses and high levels of quality and security of supply and safety" [16]. This suitable definition will be used throughout this document.

Smart grids are being deployed around the world as a direct upgrade from the traditional grid. They characterize the evolution from AMR to the new advanced metering infrastructure (AMI), which allows two-way communication enabling interaction with the meters and access to on-demand information such as various usage statistics, load profile, peak demand, outage logs and tamper notifications. The smart grid also provides better management capabilities of distributed electrical resources and enables home owners (who produce excess energy) more options when selling energy to utilities.

The European Union and the United States have been pushing forward legislation and directives to support and compel this change in their territory. The National Institute of Standards and Technology produced a conceptual model focusing on smart grid interoperability, which CENELEC later used as a basis for developing their own reference model [15, 17] for the European Union.

Since the smart meter equipment present in customer houses will have limited physical security, it is important to ensure that security mechanisms are employed to protect the

communications of smart grid networks. The imminent deployment of smart meters raises scepticism and concerns due to the potential security issues. If vulnerabilities are found after their deployment it could lead to, for example, widespread fraud, privacy breaches, or power outages in certain areas as a result of an attack. Over-regulation is also noted as one of the challenges, however, without standards the interoperability between home appliances and meters would be reduced [3].

There are already some protocols designed specifically for the smart grid. The protocol we study in this thesis is the Device Language Message Specification and Companion Specification for Energy Metering (DLMS/COSEM). This protocol is a standard protocol for remote interaction in the smart grid, often being deployed above power-line communication (PLC) links, and offering several advantages when compared to other protocols, including encoding and additional features such as push operations and digital signatures [8, 14].

The DLMS/COSEM protocol was designed following the OSI model. It is mainly an application layer protocol which provides services to establish logical connections between a client and servers. Its application layer also performs some presentation layer functions such as encoding and encryption. Some of its communication profiles also create a sublayer in the transport layer that specifies how application layer messages can be transported over various communication media [10]. DLMS models the communication entities and defines how information is formatted, transported, and processed. COSEM specifies the COSEM interface classes, the Object Identification System (OBIS), and the use of interface objects to model functions for meters.

The main reasons why this protocol became the most favoured choice within smart grid protocols were its energy-type-agnostic interface model and its clear separation between that interface model and the communication layers beneath it. For one, this allows DLMS/COSEM meters to have a very wide range of functionality, and allows both layers to evolve independently which is great due the rapid change communication protocols and media have been undertaking in the recent years.

1.1 Motivation

Electricity is a critical necessity of the modern world, and as such, its generation, distribution, and consumption need to be protected at all cost. As governments and utilities push for the modernization of the existing infrastructure, we must consider how new technologies to be inserted will be protected from attacks.

Laplante [21] provides a good set of stakeholder groups involved in the smart grid: consumer advocates, utilities, technology providers, regulators, policy makers, and environmental groups. The requirements each of them pose must be negotiated with the other groups to reach a common ground. The discussion will ultimately result in a set of re-

quirements accepted by the majority of the stakeholders. These requirements must be met in order to achieve the best solution, that will naturally gain popularity and acceptance from the industry and the consumers.

This has already happened in some countries, for example, Dutch utility providers have specified the Dutch Smart Meter Requirements (DSMR) which provides the specific requirements for Netherlands' stakeholders [21]. DSMR is accompanied by four companion standards which define in detail the functional and security requirements of a set of four ports that must be present in smart meters deployed in the Netherlands. Its companion port P3 is implemented based on DLMS/COSEM protocol [22]. At the moment, the fifth major iteration of these requirements is being rolled out.

This thesis focuses on evaluating the security of the DLMS/COSEM protocol. Ensuring the security on the communication layer of the smart grid is crucial since this is the layer which can be most easily observed by adversaries, and the one which such adversaries will use to try to interact with smart grid equipment. The communication layer is also what enables remote attacks to be carried out, and these are the most interesting attacks as they could scale easily to have a greater negative effect, such as cutting the energy supply of an entire region. Thus, it is important that the information exchanged, its format and the cipher suite (and its use) are carefully analysed.

1.2 Objectives

Since the smart grid is already being deployed as an experiment in some countries, there are reports about the security of the AMI. Such reports define good practices for smart grid protocol implementations and explain possible generic attacks. However, although the DLMS/COSEM protocol was already included in the nation-wide regulations of some countries, there is too little amount of work examining and ascertaining the effectiveness of its security features. Validation goes along with security, since it will verify if DLMS/COSEM implementations do not act in an erroneous manner.

The main objectives of this thesis are: to understand what security problems affect the DLMS/COSEM protocol; to design a framework capable of assessing the validity and security of DLMS/COSEM protocol implementations; and finally to develop a tool using the framework designed in order to test DLMS/COSEM protocol implementations. The DLMS/COSEM implementation we will be testing in our setup will be provided by a major industry partner, ensuring our results will be of practical use in real world scenarios.

However, a few more preparations are needed to properly achieve our main objective. In a first phase, we study the protocol specification and survey the existing literature on attacks and vulnerabilities concerning the smart grid, and more specifically, in the DLMS/COSEM protocol. As our knowledge of the protocol solidifies, we also attempt to identify novel vulnerabilities.

We then look for solutions on how to evaluate the security of a DLMS/COSEM protocol implementation. We define what needs to be tested and how it should be tested to guarantee accurate results until we reach one final solution. That solution will then be implemented as a tool, which will be used to test the DLMS/COSEM protocol implementation against all the attacks we collected in the first phase of our work. The tool should be able to correctly identify problems in the implementation being tested.

1.3 Contributions

This dissertation presents the following contributions:

A summarized exposition of the available literature relevant to smart grid security, focused mainly on DLMS/COSEM, including our remarks about their relevance in face of the latest version of the DLMS/COSEM protocol specification. We include detailed explanations of known vulnerabilities and attacks, and their relevance in regard to protocol specification and individual implementation weaknesses.

This dissertation's main contribution, a complete description of the ValiDLMS framework, a framework for designing tools for DLMS/COSEM implementation validation and security auditing. The framework includes the fuzzing and verification modules. The fuzzing module works using an attack database where known attacks are stored, and generating semi-random attacks based on that database. The verifying module has several components that capture the traffic and analyse it to detect vulnerabilities as well as implementation deviations from the protocol specification.

A tool implemented using the ValiDLMS framework for proof of concept and experimentation. This tool was implemented using Wireshark, a well-known mature open-source packet analyser, and is capable of analysing DLMS/COSEM traffic captured over a PLC network of smart grid components, thus becoming the first open-source tool capable of verifying DLMS/COSEM traffic over PLC networks.

An experimental evaluation from analysing a DLMS/COSEM implementation currently being tested for a country wide deployment. Using our tool, errors in messages and security weaknesses were detected in the implementation.

A ValiDLMS and our experimental tool were submitted in a paper to the 3rd Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2018), located at the 48th International Conference on Dependable Systems and Networks (DSN-2018).

1.4 Organization

This document is organized as follows:

Chapter 2 presents relevant details and information exposed in the literature about

smart grids and DLMS/COSEM protocol, effectively summarizing vulnerabilities and attacks described, including relevant individual observations.

In Chapter 3, the ValiDLMS framework for DLMS/COSEM protocol testing tools is presented. It is explained in detail the motives behind the design choices taken and how they are useful and applicable in the pursuit of our goals.

In Chapter 4 our experimental evaluation of the tool is discussed, detailing our implementation of the tool framework we propose. We then describe and discuss the tests made using it and the results obtained.

The final chapter presents the conclusion to this dissertation. The work developed throughout this dissertation is summarized and reflected upon. Some possibilities for future work are also discussed.

Chapter 2

Context and Related Work

In this chapter we introduce the advanced metering infrastructure, and provide an overview of the DLMS/COSEM protocol, focusing a bit more on its security aspects. We then provide a compilation of the vulnerabilities and attacks found in the existing literature.

2.1 Advanced Metering Infrastructure

AMR is a technology that allows utility providers to collect data from metering equipment automatically. Consumption can be monitored without having to physically verify the meter, monitoring can be made in a timely manner to predict demand and avoid shortages, and billing is more accurate.

The AMI is the next step in metering sophistication. It enables two-way communication with smart meters, which provides more capabilities than traditional meters. The AMI includes every part of the utilities infrastructure, from its meters and data concentrators to the Head End System (HE) controlling the grid [25]. Although it can be used for any type of metering, electricity utilities are predominantly leading the push for smart grid implementation.

The AMI provides advantageous features to utilities such as remote connection management capabilities and tariff programming. These and other features enable the utility to generate an accurate load profile for their grid and also remote access to their own grid equipment. The enhanced capabilities of smart meters also give way to new features, useful to motivate customers into adhering to the smart grid. For example, customers have more options for payment together with other service features, which provide a better understanding of how the energy is being spent. Using this information, the consumers can better regulate their energy consumption.

Dynamic pricing (also known as time-of-use pricing) is one main advantage of the smart grid, where the price of the utility is different according to the load of the grid at the time. Customers are continuously informed of these dynamic prices and can choose to lower their consumption during demand peak times, effectively flattening the demand

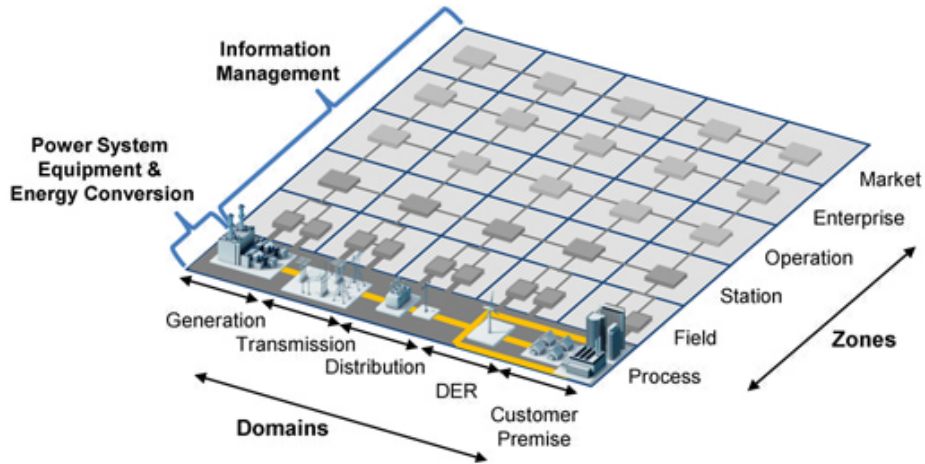


Figure 2.1: The SGAM smart grid plane [17].

during those periods while saving money and energy, both for themselves and for the grid [13].

Smart meters may also communicate with smart appliances taking advantage of dynamic pricing. Smart appliances are home appliances which can adjust their energy consumption automatically based on the current price of energy by working only during off-peak times of the day when energy is cheaper or by lowering their energy usage during peak demand when energy is more expensive. The AMI will also better accommodate distributed generation, where costumers who generate electricity may feed surplus energy to the grid.

The European Committee for Standardization, the European Committee for Electrotechnical Standardization, and the European Telecommunications Standards Institute are the responsible organizations for developing standards for smart grids in Europe. These organizations defined the Smart Grid Architecture Model (SGAM) [17] as a reference model for smart grids based on the conceptual model published by the National Institute of Standards and Technology [24]. SGAM describes multiple layers of interoperability but owing to the scope of this thesis we will only consider the communication layer (see Figure 2.1).

SGAM [17] is divided in domains and zones. Domains represent different parts of the energy conversion chain, while zones represent different hierarchical levels in the management of information used to control the domains. The smart grid plane (Figure 2.1) defined in the same technical report enables a high level view on the interactions between zones along the energy conversion chain. A service provider can be placed in any segment of the grid, depending on his role in any specific case. Zones are described in Table 2.1.

CEN-CENELEC-ESTI also produced Functional reference architecture for communications in smart metering systems [15], which defines a reference architecture focused

Zone	Description
Process	Includes equipment physically involved with the transformation of energy, e.g., generators, transformers, circuit breakers, ...
Field	Includes equipment that protects, controls and monitors the process zone, e.g., intelligent devices which process and acquire data from the power system, protection relays, ...
Station	Functions as an aggregation level for Field zones, in this zone we have, for example, neighbourhood data concentrators and monitoring systems
Operation	Hosts equipment that manages entire domains in the power system operation, e.g., AMI HE System and Supervisory Control and Data Acquisition (SCADA) systems.
Enterprise	Includes organizational processes, services and infrastructures for enterprises, e.g., asset management, billing and procurement, logistics, customer relation management, ...
Market	Systems that enable operations outside the scope of a single enterprise, e.g., energy trading, mass market, ...

Table 2.1: Simplified description of SGAM zones [17].

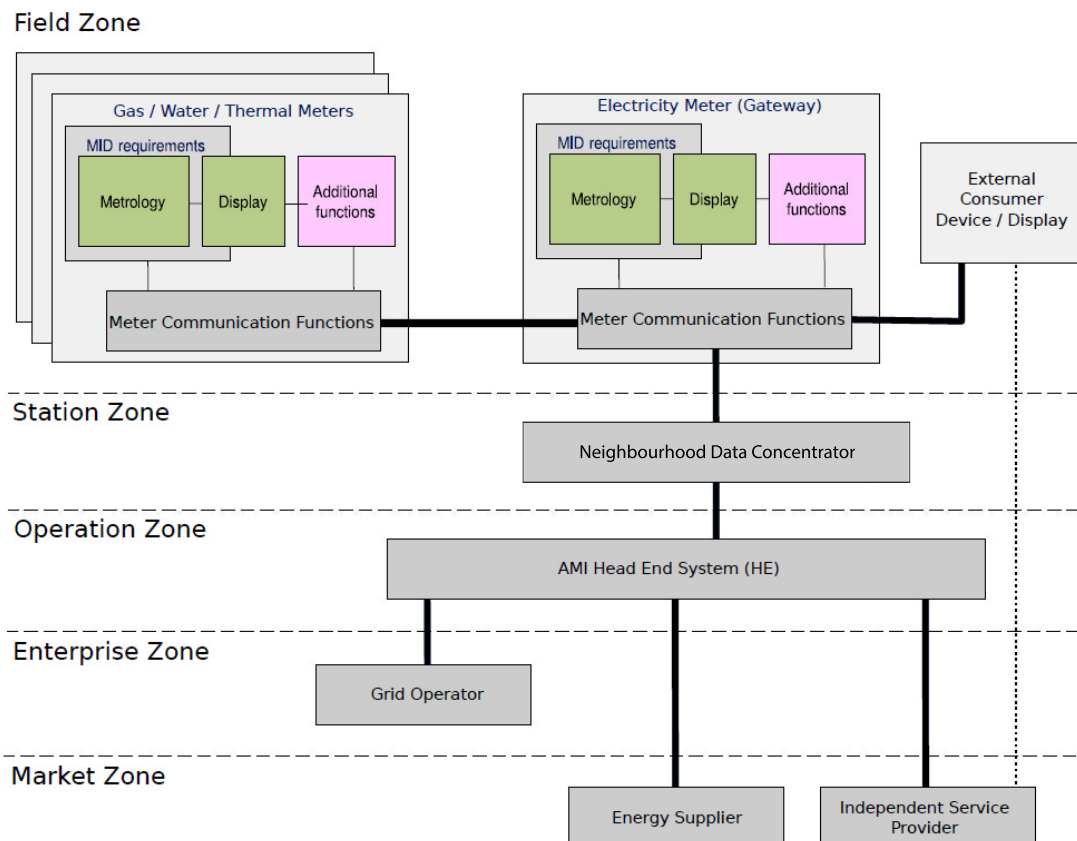


Figure 2.2: AMI simplified topology [15, 29].

on communications in the smart grid. Figure 2.2 shows an overview of some devices and their connections in a smart grid that follows this reference architecture. Optionally, the smart grid may include smaller networks. Smart meters within the same household may communicate among each other using a Home Area Network (HAN). Local networks (LN) and neighbourhood networks (NN) may be used in cases where a dedicated WAN connection to a single meter/LN is not desired. Access points for these networks have gateway functionality because they need to interconnect different networks.

The AMI HE System is the central piece in the AMI architecture. It controls and communicates with the devices in the grid, sending pricing and other useful information, monitoring the system for device tampering, and so on. They provide the top interface to smart grid functions and applications.

2.2 DLMS/COSEM Protocol

2.2.1 Overview

The DLMS/COSEM protocol is comprised by the Device Language Message Specification and the Companion Specification for Energy Metering. DLMS models the communication entities and defines how information is formatted, transported and processed. COSEM specifies the COSEM interface classes, the OBIS identification system and the use of interface objects to model functions for meters.

The DLMS/COSEM specification is documented and maintained by the DLMS User Association, in the so-called "coloured books". The Blue Book[9] specifies the COSEM object model and OBIS system, the Green Book[10] specifies the communication model and process, the Yellow Book describes the conformance testing process of equipment implementing the DLMS/COSEM protocol, and the White Book compiles a glossary of DLMS/COSEM terms.

Excerpts of the first three books are available at the DLMS UA website [11], however, these versions have been heavily abridged. The full versions of these documents are only available to members of the DLMS UA.

In DLMS/COSEM, a physical device (e.g., meter) holds one or more logical devices and each logical device contains a set of COSEM objects. Each logical device is bound to a Service Access Point (SAP). SAPs depend on the layer supporting the COSEM application layer, which is determined by the communication profile being used. COSEM objects are objects that instantiate the interface classes defined in the Blue Book. These objects have attributes and methods, and represent things like clocks, credit, associations, and setups.

The DLMS/COSEM application layer has one main component: the Application Service Object (ASO). This component provides services to its service user and other COSEM application processes (APs), and uses services provided by the underlying layer.

The ASO can be divided in three mandatory components, offering one additional optional component for client APs. The three mandatory components are: the Association Control Service Element (ACSE) which provides services for establishing and releasing application associations, the extended DLMS Application Service Element (xDLMS ASE) which provides services to transport data between COSEM APs, and the Control Function (CF) which specifies how the ASO will invoke the appropriate service primitives of the other components.

The additional optional component for clients is a client Short Name (SN) mapper which provides a mapping between services using Logical Name (LN) and SN referencing. This component is present when the server uses SN referencing, which uses only two bytes. In LN referencing, COSEM object attributes and methods are referenced by the logical name of the instance they belong to, using OBIS (OBject Identification System) codes. OBIS codes consist of a group of six bytes, and are defined in the Blue Book. In SN referencing, COSEM object attributes and methods are mapped to DLMS named variables. There are different xDLMS ASE service sets for each type of referencing. This component allow client applications to be unconcerned with the referencing type used by the server.

The Blue book defines COSEM Interface Classes (IC) as well, which work like interfaces in the Java language. COSEM objects instantiate ICs, each having their own attributes and methods. Different levels of access and visibility will be set depending on the application association established between the client and the server. Each IC is mapped to an unique class id, which identifies it.

The latest version of DLMS/COSEM defines four communication profiles: 3-layer, connection-oriented HDLC; TCP-UDP/IP; S-FSK PLC; and wired/wireless M-Bus. Using one of these profiles, a client can connect to a server.

The client first connects to the Management Logical Device on the server using the public client Application Process (AP). The existence of this logical device on every server is mandatory and it always occupies the first logical device address. It allows the connection of public client APs using Lowest Security Authentication (no security). It then reads the "SAP assignment" object which reveals the logical devices present in that physical device.

The client is then able to establish an Application Association (AA) with the server's logical device via the ACSE protocol, negotiating his requirements and communication details, and creating an association object. This association object contains the application context, the xDLMS context, the authentication mechanism, and more. After setting up this object, the client and the server may perform authentication. The client may send his requests and obtain responses, and release the AA. This is illustrated in the central area of Figure 2.3.

The abstract definition of COSEM APDUs (Application Protocol Data Unit) is spec-

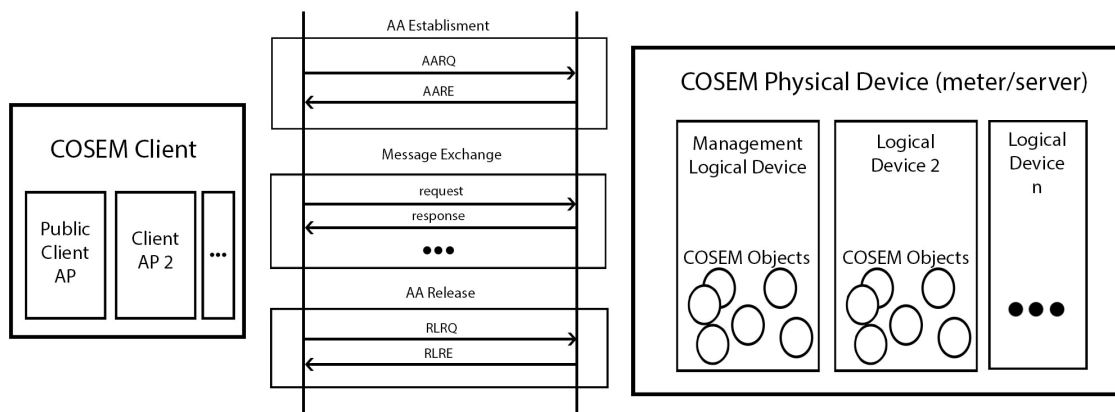


Figure 2.3: DLMS/COSEM simplified model and communication pattern.

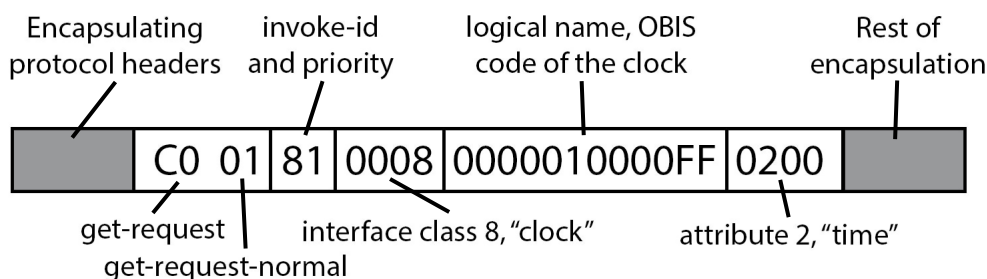


Figure 2.4: Encoded DLMS APDU used to get the time attribute from a clock object.

ified in [10] using Abstract Syntax Notation 1 (ASN.1). The same manual provides a mapping from the ASN.1 definition to XML Schema Definition (XSD) for COSEM XML representation [5].

In terms of encoding, ACSE APDUs are encoded in BER [2] and xDLMS APDUs are encoded in A-XDR. The Adapted eXternal Data Representation (A-XDR) encoding rules are a special-purpose set of encoding rules defined specifically for optimizing DLMS PDUs [1]. Additionally, push notifications can be set up to encode the DataNotification APDU in XML using the COSEM XML representation.

We present encoded APDUs in Figures 2.4 and 2.5, to illustrate how encoded DLMS messages would be used to get the time from a server. The structure of the message is defined in the aforementioned ASN.1 specification, and the semantic meaning of the values can be consulted in the Blue book.

2.2.2 Security

Due to the nature of the information monitored and gathered by the grid, the DLMS/COS-EM specification includes several security mechanisms. These mechanisms are implemented in the DLMS/COSEM application layer and they can be used in any communica-

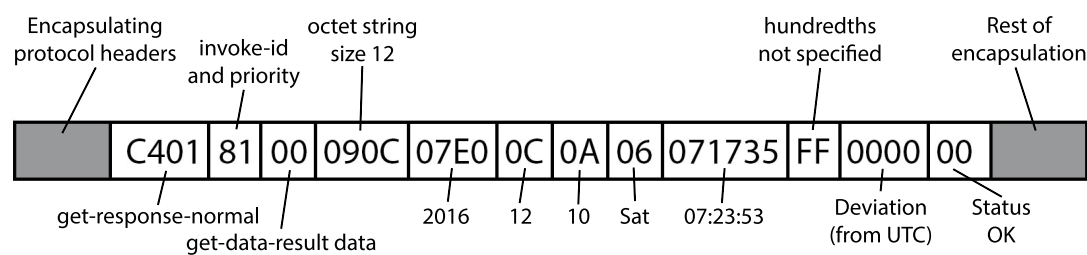


Figure 2.5: Encoded DLMS APDU response for the one in Figure 2.4.

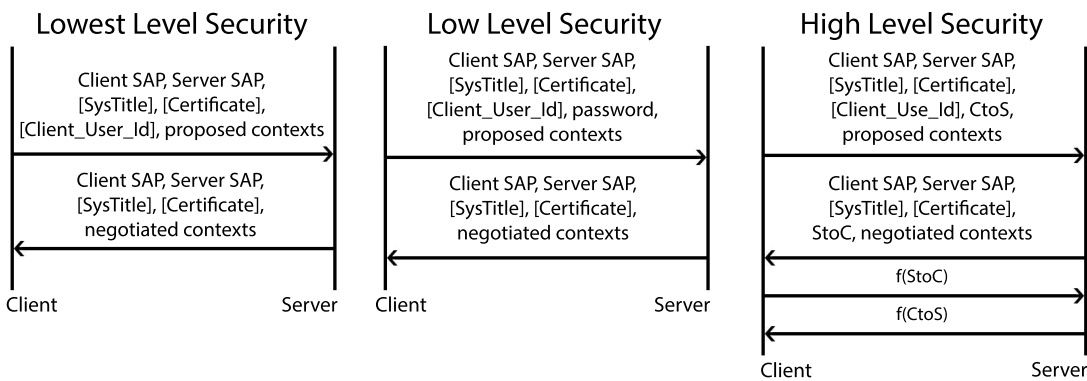


Figure 2.6: Overview of the levels of authentication supported by DLMS/COSEM.

tion profile.

During the AA establishment phase, the server and the client may authenticate with each other. After an AA between a client and a server has been established, that client’s interaction with COSEM objects in the server is subject to the security context and access rights of the AA. Furthermore, xDLMS APDUs and the COSEM data carried by them can be cryptographically protected. End-to-end security may also be configured to allow third-parties to use a client as a broker, in order to reach a server.

DLMS/COSEM supports three levels of authentication mechanisms: lowest level security, low level security (LLS), and High Level Security (HLS). Figure 2.6 shows an overview of the DLMS/COSEM authentication mechanisms. Lowest level security authentication simply bypasses authentication and is intended to allow the client to get only basic information about the server. LLS authentication enables the client to authenticate with the server by providing a password known to the server. HLS allows authentication of both the server and the client of an AA, by means of issuing challenges and verifying responses over four communication steps. HLS authentication may use one of five available mechanisms, presented in Table 2.2. The use of HLS MD5 and HLS SHA-1 mechanisms are discouraged in the latest version of the protocol specification, due to their vulnerability to collision attacks.

Security contexts and access rights for AAs are pre-configured in the server. Objects

Authentication mechanism	Step 1 $C \rightarrow S$	Step 2 $S \rightarrow C$	Step 3 $C \rightarrow S$	Step 4 $S \rightarrow C$
HLS Man. Spec.	CtoS (8-64 bytes)	StoC (8-64 bytes)	Manufacturer Specific	Manufacturer Specific
HLS MD5			MD5(StoC HLS Secret)	MD5(CtoS HLS Secret)
HLS SHA-1			SHA-1(StoC HLS Secret)	SHA-1(CtoS HLS Secret)
HLS GMAC	CtoS, SysTitle-C (optional)	StoC, SysTitle-S (optional)	SC IC GMAC (SC AK StoC)	SC IC GMAC (SC AK CtoS)
HLS SHA-256			SHA-256 (HLS_Secret SysTitle-C SysTitle-S StoC CtoS)	SHA-256 (HLS_Secret SysTitle-S SysTitle-C CtoS StoC)
HLS ECDSA	CtoS, SysTitle-C (optional), Certificate-Signed-C (optional)	StoC, SysTitle-C (optional), Certificate-Signed-C (optional)	ECDSA (SysTitle-C SysTitle-S StoC CtoS)	ECDSA (SysTitle-S SysTitle-C CtoS StoC)
Legend: C - Client; S - Server; CtoS, StoC - Challenge from client to server, and vice versa; IC - Invocation Counter; AK - Authentication Key				

Table 2.2: Available HLS authentication mechanisms [10].

that model AAs contain reference to the security context of the AA. The security context holds the security suite, the security policy, and other security material such as keys, initialization vectors, and certificates.

The security policy defines if authentication, encryption, or digital signature are required for request or response APDUs. Access rights determine access to COSEM attributes and methods, and the protection to be applied to APDUs that access particular COSEM attributes and methods. The protection applied to APDUs will be the stronger requirement between security policy and access rights. Moreover, APDUs with stronger protection than required are always allowed, while APDUs with weaker protection than required are rejected. The security policy can be set to require any or any combination of the following: authenticated requests, encrypted requests, digitally signed requests, authenticated responses, encrypted responses, or digitally signed responses. Access rights define the previous requirements, plus read-access and write-access to attributes, and access to methods.

COSEM objects present in servers can be protected through "Data Protection" objects, which provide indirect access to attributes and methods while applying, verifying and removing cryptographic protection.

Security Suite ID	Authenticated Encryption	Digital Signature	Key Agreement	Hash	Key Transport	Compression
0	AES-GCM-128	-	-	-	AES-128 key wrap	-
1	AES-GCM-128	ECDSA P-256	ECDH P-256	SHA-256	AES-128 key wrap	V.44
2	AES-GCM-256	ECDSA P-384	ECDH P-384	SHA-384	AES-256 key wrap	V.44

Table 2.3: Available security suites [10].

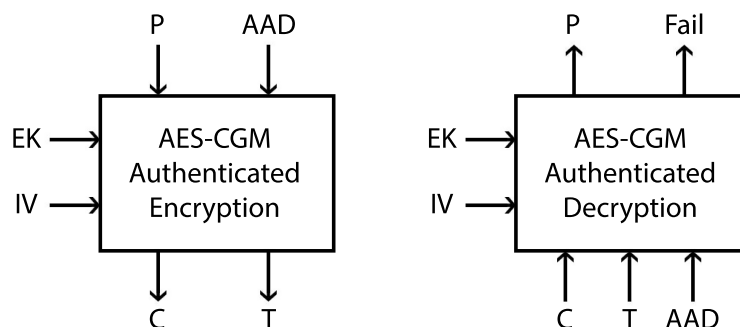


Figure 2.7: Overview of the inputs and outputs of GCM [10, 12].

DLMS/COSEM provides three security suites for use in cryptographic protection, each specifying the algorithms for authentication, encryption, key agreement, digital signature and hashing. Table 2.3 shows the currently available security suites.

DLMS/COSEM uses symmetric cryptography for authentication in HLS authentication mechanisms, authentication and encryption of xDLMS messages, and authentication and encryption of COSEM data. For these purposes, DLMS/COSEM uses the Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) to provide authenticated encryption. This mode of operation can protect authenticity and/or confidentiality. The GCM authenticated encryption operation takes as inputs: plaintext P , additional authenticated data AAD , an encryption key EK , and an invocation vector IV . The additional authenticated data consists of the security control byte and authentication key in most cases. The invocation vector includes the system title and invocation counter. The operation produces a ciphertext C if encryption is required, and/or an authentication tag T , if authentication is required. Authentication tags are twelve bytes long. The GCM authenticated decryption operation takes as inputs: ciphertext, authentication tag, additional authentication data, encryption key, and invocation vector. If the decryption process was successful, a plaintext correspondent to the input ciphertext is produced. If this process fails, then the ciphertext has been either accidentally or intentionally modified, and GCM can detect both cases.

GCM and GMAC are described in the "Recommendation for Block Cipher Modes of

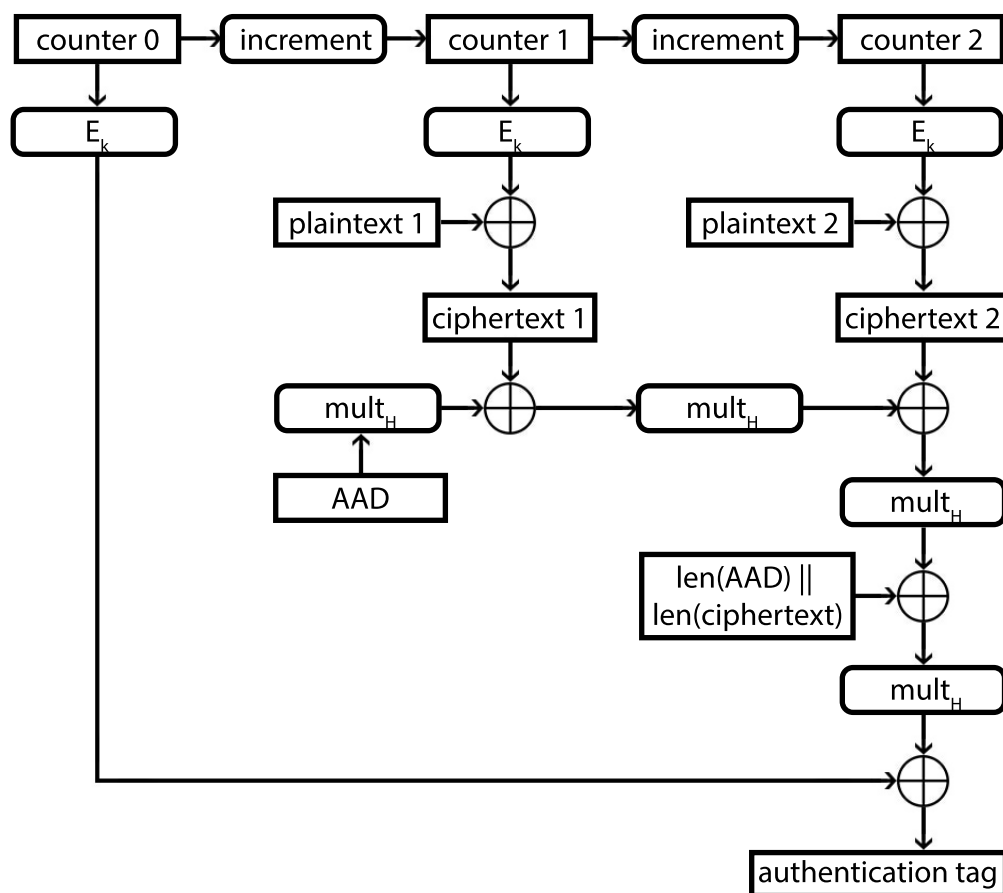


Figure 2.8: Overview of GCM authenticated encryption [12].

Operation: Galois/Counter Mode (GCM) and GMAC”, published by NIST [12]. Figure 2.7 illustrates the input and outputs of GCM operation, and figure 2.8 details the GCM authenticated encryption operation.

Between each DLMS/COSEM client-server pair, a master key and global keys are established. The master key is used as a key encryption key to encrypt other key material, and may be established through key wrap, key agreement, or out of band. Global keys are used over several AAs established between the same client-server pair, and may be a unicast encryption key, a broadcast encryption key, or an authentication key. Global encryption keys are used to encrypt APDUs and COSEM data, and authentication keys are used in the AAD input of the GCM authenticated encryption and decryption operations if authentication is enabled. Dedicated unicast keys may be used to encrypt communication between parties during an AA. Dedicated keys are transported during the establishment of the AA in which they will be used. Ephemeral keys may be used for single exchanges within an AA.

Asymmetric cryptography is used in DLMS/COSEM to authenticate communicating entities, to digitally sign xDLMS APDUs and COSEM data, and to perform key agree-

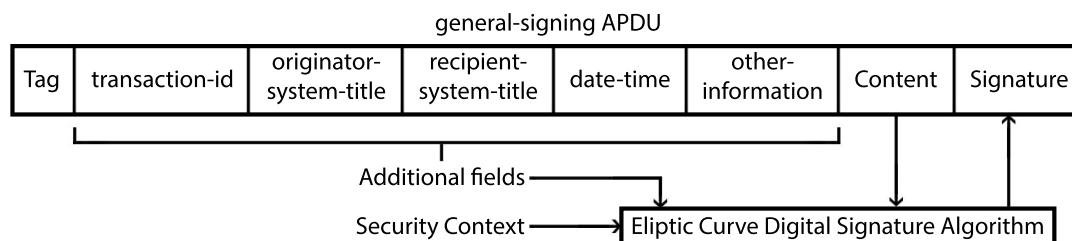


Figure 2.9: Structure of general-signing APDUs in DLMS/COSEM [10].

ment. DLMS/COSEM uses elliptic curve algorithms due to their suitability for embedded devices, as they offer the same security level when compared with RSA, but with reduced key sizes. In DLMS/COSEM, the Elliptic Curve Digital Signature Algorithm (ECDSA) is used to generate digital signatures using NIST Curves P-256 and P-384 [20]. Digitally signed data is protected against authenticity, integrity, and non-repudiation, as long as the private key remains secret.

DLMS/COSEM uses X.509 version 3 certificates, with extensions as defined in the Green Book [10]. Three types of certificates for end entities: digital signature certificate, static key agreement key certificate, and an optional TLS certificate. Trust anchors are first placed in servers during the manufacturing process using a trusted out of band process. DLMS/COSEM offers methods to provide the server with further certificates, to export certificates from servers to clients, to generate certificate requests that can be used to get new certificates from Certification Authorities (CA), and to remove certificates from servers. Figure 2.9 shows the structure of a general-signing APDU, showing also which parts of information contribute to the digital signature process.

Key agreement allows two entities to create a shared key, such that the created key remains a secret to eavesdroppers. This allows the two entities to communicate secretly afterwards using the established key. DLMS/COSEM provides three schemes of elliptic curve key agreement: the ephemeral unified model scheme, the one-pass Diffie-Hellman scheme, and the static unified model scheme. To use these, both parties must have a copy of the exact same set of domain parameters. These domain parameters holds the curve parameters such as cofactor, order and coordinates of its base point, and so on. The ephemeral unified model is used to agree on the master key, on global encryption keys, and on the authentication key. The one-pass Diffie-Hellman and static unified models are used to agree on ephemeral encryption keys, that will be used to protect xDLMS APDUs and COSEM data. These key agreement schemes work as shown in Figure 2.10. After the exchange shown, both entities apply the Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) primitive to compute a shared secret Z , which is then used together the other information to derive a key, by applying the NIST Concatenation Key Derivation Function (KDF) [6]. This other information provided to the function defines how and for

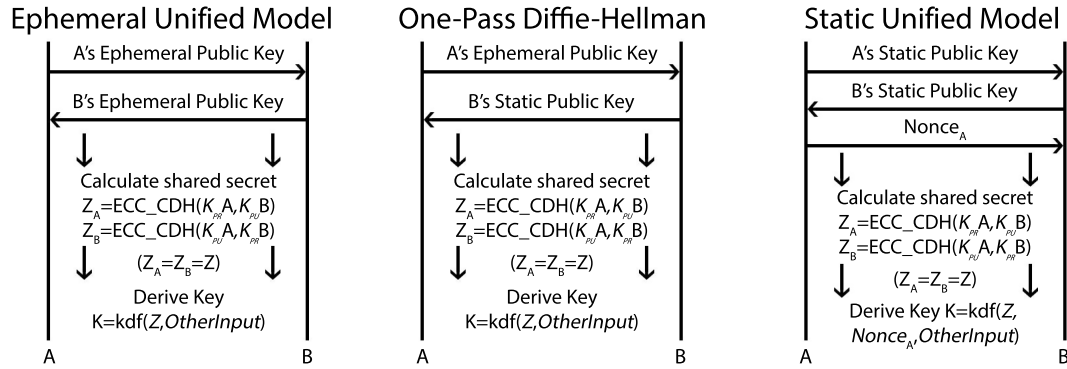


Figure 2.10: Overview of the elliptic curve key agreement schemes present in DLMS/COSEM [6, 10].

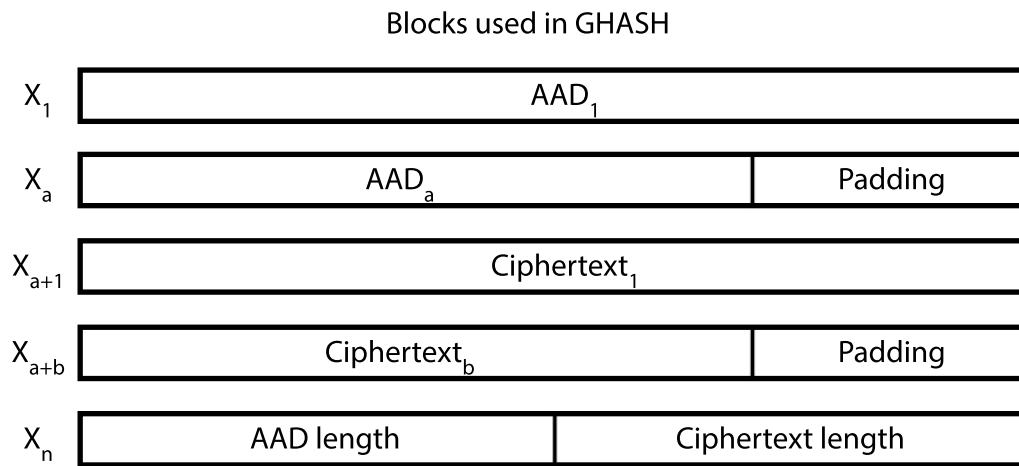


Figure 2.11: Overview of block structure of the authentication data used in GHASH to create authentication tags when also using encryption [29].

what purpose the key will be generated, and the public identifiers of the parties involved. Static keys are obtained in a trusted manner (e.g., from certificates signed by trusted certificate authorities).

The authentication tag is generated through the concatenation of additional authentication data, ciphertext, and their respective lengths. The concatenation uses padding if necessary at the end of each component to organize them in 128-bit blocks. The end result X is shown in figure 2.11. Using $H = \text{EncryptAES}(\text{key}, 0^{128})$ and the previous result X , a polynomial Y is calculated as such:

$$Y = \sum_{i=1}^n X_i \times H^i$$

The authentication tag is afterwards generated as: $T = Y_n + E_K(\text{IV} || O^{31} || 1)$.

When protecting APDUs, the kind of protected APDU produced depends on the

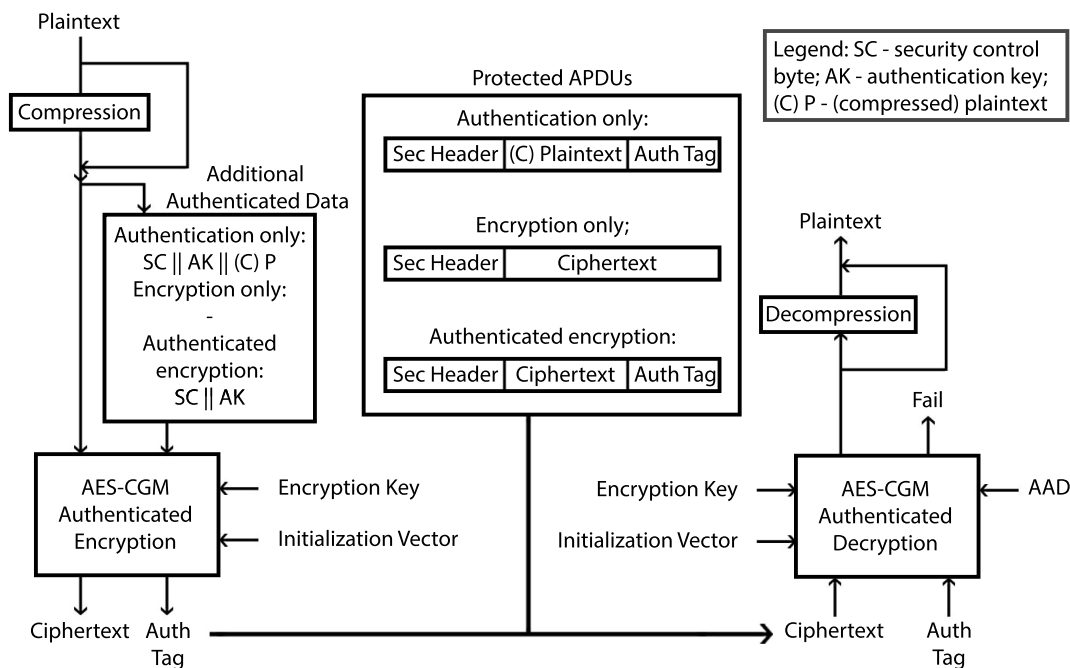


Figure 2.12: Overview of the algorithm used for protecting APDUs in DLMS/COSEM [10].

xDLMS service being protected. Certain xDLMS APDUs use service-specific ciphering and have two ciphered APDU variants available, one using the global key, and one using the dedicated key. Other xDLMS APDUs use general global ciphering or general dedicated ciphering APDUs. There is also a general-ciphering APDU which carries the necessary information on the key to be used. The general-ciphering APDU may also be used between third parties and servers.

These protected APDUs include a security header which is composed by the security control byte and the invocation counter. The security control byte indicates the security suite being used, whether or not authentication or encryption are applied, if the key set being used is global or dedicated, and if compression is being used. The AAD used in the GCM authenticated encryption and decryption operations differs according to the protection applied to APDUs. In authenticated APDUs, the AAD consists of the security control byte, the authentication key, and the information being authenticated in compressed format. In encrypted and authenticated APDUs, the AAD consists of the security control byte and the authentication key. General-ciphering APDUs contain additional information in the AAD used. Figure 2.12 shows the algorithm used to protect APDUs.

2.2.3 Vulnerabilities

In this section we present the vulnerabilities relevant to DLMS/COSEM found in the literature. We make a distinction between practical vulnerabilities and theoretical vul-

nerabilities. Practical vulnerabilities can be exploited to attack the security of a system. Theoretical vulnerabilities cannot be exploited yet due to technological restraints, despite the recent incidents of cryptographic hash functions have been showing of how the constant evolution of hardware and cryptanalysis techniques turn theoretical vulnerabilities into practical ones. For example, Stevens [26] could achieve a freestart collision on SHA-1 recently, claiming the first practical break of the full SHA-1. Hence, we believe theoretical vulnerabilities have a high probability of being exploited in the near future.

There is very little research published about the security of DLMS/COSEM. Most of the vulnerabilities exposed in this section were exposed by Weith [29], in the most complete security analysis of the DLMS/COSEM protocol available to date.

2.2.3.1 Protocol Vulnerabilities

This subsection lists vulnerabilities which originated from the protocol specification. It is important to note that these vulnerabilities were found in a version of the DLMS/COSEM protocol prior to the latest version available which adds a lot of security related mechanisms.

Removable authentication The security header is sent in plaintext in protected APDUs, which makes it prone to modification. By altering one bit in the security header, removing the last twelve bytes of an APDU, and updating its length, an attacker can transform an authenticated and encrypted APDU into an encrypted only APDU. If the security policy of the target device allows encrypted only APDUs, the attacker can downgrade the protection applied in APDUs, effectively resulting in a downgrade attack. It is also important to note that, since GCM encryption is a counter-based cipher, APDUs protected using it can be specially vulnerable to bit flipping attacks. Removing authentication removes the ability to detect the alteration of an APDU. Weith [29] claims to have successfully exploited this vulnerability in their practical setup. Having reviewed the details relevant to this attack in the specification, we have confirmed that this has not been addressed in its latest version.

Information leakage Some xDLMS services use service-specific ciphering that results in a ciphered variant of the normal xDLMS APDU. In this case, the tag which identifies the APDU type is left in plaintext, revealing which xDLMS service is being transported in the APDU. For example, a get-request APDU has two ciphered counterparts: glo-get-request and ded-get-request, for encryption using global or dedicated keys, respectively. This makes it trivial for an attack to know when a get-request has been sent, regardless of APDU protection. Furthermore, Weith [29] claims that this information is leaked unnecessarily, due to observing that this information was just used to determine how to decrypt the APDU. This could have been avoided by using general ciphering, instead

of service-specific ciphering. In his setup, Weith experienced no change in packet processing when he altered a message type from `glo-get-request` to `glo-get-response`, which further supports this conclusion. The fixed message sizes can also help an observer discern which messages are being sent, especially when a message with an unique size is used. This helps in guessing plaintexts for those messages, which in turn is useful for use in known-plaintext attacks. Additionally, a lot of known plaintext can be gathered from other sources, such as the User-Information fields in AARQ responses which provide 15 bytes of known plaintext. [29] found that HLS 5 authentication responses were the most reliable source of known plaintext, providing 22 bytes of known plaintext.

HLS server impersonation According to [29], a malicious server could impersonate a legitimate server via replay of the client's title, nonce, and hashed responses. This is said to be only plausible during a plaintext session since it requires the attacker to be able to read and respond to the client we would fake our authentication with. HLS requires mutual authentication between parties, using four steps for that. In the first two the parties exchange a challenge, and in the next two steps, the parties exchange the hashes of the challenge and other info received (depending on the method chosen) concatenated with a shared secret (HLS secret). The attacker would need to replay the answer to a nonce that was previously used, which is very unlikely assuming nonce generation is secure.

HLS offline dictionary attack If an attacker can sniff the authentication packets exchanged during HLS authentication, he can read the challenges and their respective hashes. Knowing that the HLS MD5 and SHA-1 mechanisms produce hashes as $h(\text{challenge} \parallel \text{HLS_secret})$, an attacker may carry an offline dictionary attack by attempting to reproduce the hash using candidates for `HLS_secret` until he can successfully generate it. He will then know the `HLS_secret` used for between this client-server pair and this method of authentication will be unsafe from there on.

Responses not tied to requests There is nothing included in responses that allows a client to link the request he sent to the response he got. The client cannot differentiate between the real response and a valid response introduced by a possible attacker in a MITM attack.

MAC Forgery without Authentication Key Knowing that AAD blocks are fixed across multiple messages, and having access to the encryption key K , and one ciphered and authenticated message, an attacker can forge an authentication tag without knowledge of the authentication key. This forgery is limited as the forged ciphertext must have be divisible into the same number of 128-bit blocks. In a simplified manner, the authentication

tag is calculated as:

$$T = K_0 + \sum_{i=1}^n X_i \times H^{n-i+1}$$

Since $GF(2^{128})$ is commutative, and the elements are polynomials over \mathbb{Z}_2 , the order of additions does not matter, moreover, addition and subtraction are the same operation. We then can observe that for an example with two AAD blocks and one ciphertext block we have:

$$T = A_1H^4 + A_2H^3 + C_1H^2 + X_nH + K_0$$

The attacker first calculates H and K_0 using the key K and a known X . Next, he generates a forged ciphertext C' and an updated X_n to account for length variations in the new forged blocks. It is then possible to substitute the legitimate parts with the forged ones because they can be subtracted as such:

$$T = A_1H^4 + A_2H^3 + C_1H^2 + X_nH + K_0 \equiv$$

$$T + C_1H^2 + X_nH = A_1H^4 + A_2H^3 + K_0 = S$$

Finally, the attacker is able to forge an authentication tag:

$$T' = S + C'_1H^2 + X'_n = A_1H^4 + A_2H^3 + C'_1H^2 + X'_nH + K_0$$

The authentication tag produced looks legitimate and was generated without any knowledge of the authentication key. This is a vulnerability because the authentication key was not required to authenticate a message.

Plaintext APDUs accepted in a ciphered context According to the protocol specification, APDUs sent in a ciphered application context are accepted even if they contain plaintext messages and have encryption disabled in their security header. If a ciphered type APDU reaches the server with the encryption and authentication disabled in its security header and plaintext content, it should not be parsed as if it was ciphered because it is malformed and likely to have been sent by an attacker. Implementations should be able to recognize this pattern and be resilient to it.

2.2.3.2 Potential Vulnerabilities in Implementation

This subsection lists vulnerabilities which could result from errors in implementing the protocol, and do not derive from weaknesses in the protocol specification.

Lack of Replay Protection If servers accept frame counters equal or lower than the expected frame counter for a certain device, they are left vulnerable to replay attacks and re-usage of certain elements that should not be re-used like the invocation counter.

Predictable Nonces This enables challenge responses to be replayed if the attacker observed the nonce being used previously and captured the correspondent response packet. Nonces should be unique and avoid repetition to be secure.

Identical AA Titles Allowed Allowing identical AA titles would let an attacker use the HLS server impersonation attack. If an identical AP title is used when trying to establish an AA, this should be denied by default, by the server (responding AP).

Cipher-Only APDU's Accepted The keystream can be recovered using a ciphertext and a known plaintext. If the application using GCM ignores the ICV (Integrity Check Value, acts like an incremental MAC), it is possible to create a forged message using this keystream and no authentication, and such message would be undetectable. The replacement message is, however, limited in that it cannot exceed the length of the original message (due to the retrieved keystream's length).

Plaintext AARQ Elicits Ciphertext AARE A plaintext AARQ could be used to obtain an encrypted AARE from the server. If paired with the allowance of identical AA titles and the acceptance of cipher-only APDUs, described above in this section, an attacker who knows the LLS password can obtain the necessary data to allow him to generate an AARQ which will be accepted by the server.

Plaintext Authentication Allowed LLS authentication is completely vulnerable to passive interception, allowing for MITM attacks. HLS authentication using proprietary, MD5 or SHA-1 hashes are susceptible to the HLS server impersonation attack.

On-Line Dictionary Attack In [29], a different behaviour in the metering equipment between the response of inputting a wrong password and inputting a correct password was noticed, even after failing and disallowing login. The error message was even different. It is noted that if timing is also different in this case, then an online dictionary attack can also take place.

Insecure HLS Authentication Supported Bad regulations in some countries have forced grid operators to support it along with other insecure HLS security suites. All implementations that follow regulations that similarly enforce support of insecure mechanisms hinder the efforts of securing the grid and allow for exploitation of said mechanisms.

Arbitrary Client Titles Accepted The invocation counter is reset to 0 when the key is established. When the authenticated encryption function is used, the IC is used and then incremented by 1. The device maintains a data structure with the system titles and the

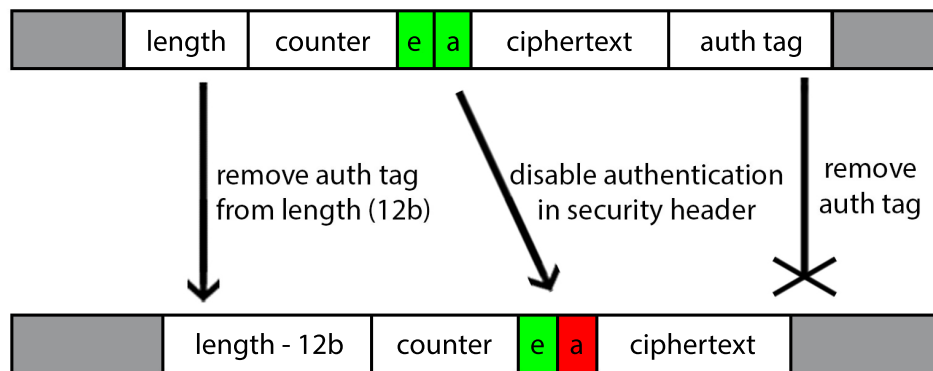


Figure 2.13: Procedure used to remove authentication from APDUs [29].

last IC used for their last authenticated interaction. If clients are allowed to connect using any system title, it will very likely reset the IC of the corresponding AA. The memory of the device might also be exhausted by an attacker, by filling it with dummy system titles. This results in the meter erasing legitimate system titles. The attacker will then be able to connect using one of those previous legitimate system titles by replaying messages in order from the beginning, due to the IC always starting at 0 and due to its incremental nature. A possible way to thwart this replay attack is to configure servers to allow only certain system titles to connect with it.

Removable Authenticators It is possible to remove the authenticator from an authenticated APDU. The procedure to perform this action is shown in 2.13. First, the attacker removes the last 12 bytes effectively erasing the authentication tag from the APDU, then simply readjusts the length by subtracting the removed bytes, and lastly disables the security control byte to encryption-only. This is also a good reason why cipher-only APDUs should be rejected. Figure 2.13 shows the procedure described.

2.2.4 Attacks

In this section we expose some of the attacks described in [29], that use the vulnerabilities described in the previous section.

2.2.4.1 Message Replacement

If cipher-only messages are accepted, an attacker can replace a legitimate ciphertext with a malicious ciphertext generated from a plaintext of his choosing, as long as he knows or guesses the plaintext of the legitimate ciphertext. The keystream used to cipher the message can be recovered by applying the XOR operation on the known plaintext and the corresponding ciphertext, as such: $C = P \oplus K_s \rightarrow C \oplus P = K_s$. The recovered keystream can now be used to cipher a plaintext of the attacker's choosing, since the legitimate IV

can be used and no anomaly will be detected under normal conditions. If the message containing the legitimate ciphertext is authenticated the replacement could be detected, therefore the authenticator must be removed before the message is sent.

2.2.4.2 Command Injection Proxy

This attack depends on the lack of replay protection and on the server accepting cipher-only APDUs. A proxy is mounted between a legitimate server and a legitimate client. This proxy forwards messages unchanged except for the client's HLS response message. This message is used to compute the keystream using the previous attack and stores it along with the title and IV. The connection then proceeds as normal and unchanged. After the connection is terminated, the attacker can re-use the keystream to connect with the server and send forged messages in order to execute commands, including disabling security mechanisms or setting passwords for other authentication mechanisms not used by the client. That way the attacker gets prolonged access to the server without interfering with normal access.

Chapter 3

The ValiDLMS Framework

As stated previously, there are very few or no tools (depending on the type of communication) available for ascertaining the correctness and security of the DLMS/COSEM protocol. Dantas developed eFuzz [7], a fuzzer for testing bugs with security implications in DLMS/COSEM implementations. The fuzzing capabilities described provide an automated way to test thousands of input combinations within a time-frame short enough to motivate its use. eFuzz has several shortcomings that overlap with those of other fuzzers, such as detecting correct user permissions or exposing multi-stage vulnerabilities. There are also no known open-source tools for testing DLMS/COSEM over power-line communication (PLC) at the time of writing. A few factors may contribute to this situation, namely the access to the actual components (e.g., DTCs and meters) and the unavailability of open-source libraries for PLC. However, as DLMS/COSEM is being deployed in the power grid, there is the need to verify if the operation of this protocol over PLC is being done in an appropriate and secure manner. This problem is exacerbated by many smart grid enterprises acquiring their DLMS/COSEM implementations from third-parties.

In the following sections of this chapter we detail ValiDLMS, our framework for developing future DLMS conformance and security testing tools, designed based on what was observed in the previous chapters. In the first section, a high-level view of the entire framework is presented in order to understand how components interact with one another and where they are located. The following section presents the interaction layer and explains its individual components. Then, the testing layer is broken down in modules and components which are explained individually in their respective subsections.

3.1 Framework Overview

ValiDLMS is an architecture framework for developing tools that test DLMS/COSEM implementations for conformance and security. The ValiDLMS architecture attempts to be as complete as possible, providing capabilities for validating DLMS/COSEM protocol implementations, and better security analysis through the use of fuzzing techniques and

customizable vulnerability tests. ValiDLMS respects the separation made by the standard between media and model that allows for independent evolution and also allows for that to happen in the framework.

The architecture of ValiDLMS is displayed in Figure 3.1. It is divided in three layers: *DLMS/COSEM environment*, *interaction*, and *testing*.

In a typical deployment scenario of DLMS/COSEM, a client data concentrator communicates with the smart meter servers to collect and/or set local information. One would like to determine if the exchanged messages (DLMS/COSEM environment layer) follow the specification of the protocol and if they are properly secured (Testing layer). The interaction layer interconnects these two layers, allowing the transmission of information between them. This means that, on one hand, the interaction layer captures the DLMS/COSEM communications occurring in the environment, and then delivers them to the testing layer. Here, they will be analysed by a tool to determine if the messages conform to the protocol specification and if the environment is protected from attacks, warning users when any anomaly is found. On the other hand, the interaction layer is also used as a means for the tool to perform security tests against the DLMS/COSEM environment, by carrying out attacks via packet injection with a fuzzer. Furthermore, attacks executed by the tool should also be captured, any effects should be detected, and warnings should be issued should any security deviations be found.

The DLMS/COSEM environment layer contains the implementation and equipment being tested, and may be composed of many possible sets of meter servers and data concentrating clients.

The interaction layer links the DLMS/COSEM environment layer to the testing layers, providing full two way communication for the fuzzing module, and packet sniffing for the verification module of the testing layer.

The testing layer comprises the core tool of the framework. This tool has its components divided in two modules: fuzzing and verification. The fuzzing module is capable of generating semi-random attacks according to an attack database. The verification module verifies any traffic captured from the DLMS/COSEM environment layer through the interaction layer for conformity with the specification, weaknesses, and attacks.

3.2 Interaction layer

The interaction layer is composed by two components, namely the *DLMS client* and the *sniffer*. These support interactions between the testing layer which comprises most of the tools, and the DLMS/COSEM environment.

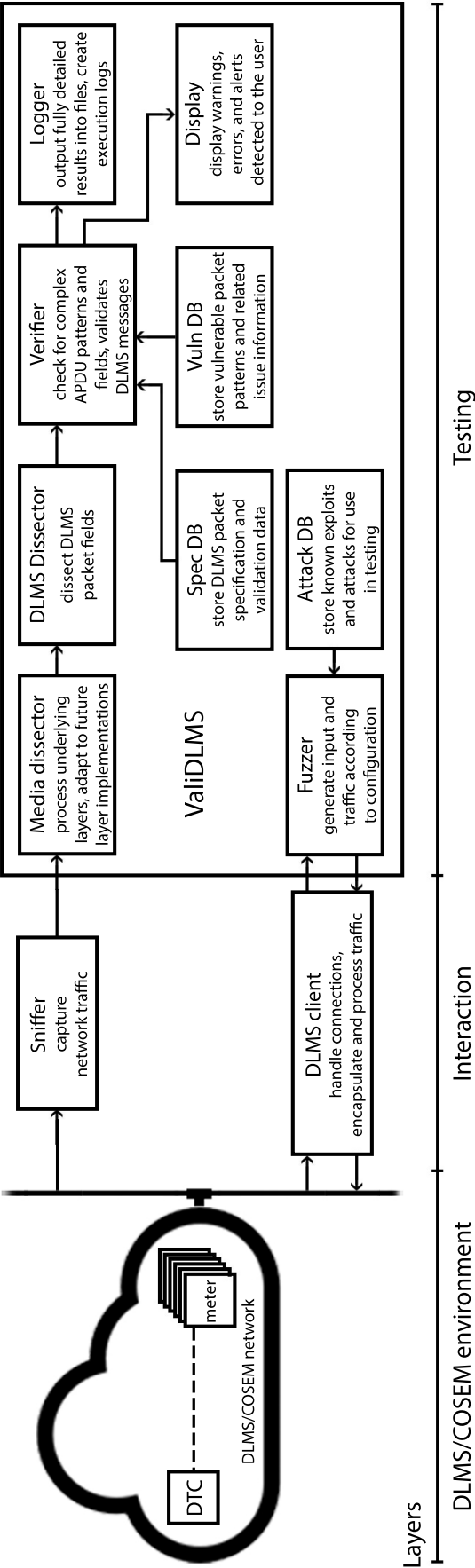


Figure 3.1: Overview of the ValiDLMS architecture for DLMS/COSEM testing tools.

3.2.1 DLMS Client

The DLMS client component enables a two way communication between the fuzzing module of the tool and the DLMS/COSEM environment. It manages all the necessary connections, allowing the tool to take part in complex interactions with a DLMS/COSEM network, letting the fuzzing module perform multi-stage attacks and other sophisticated malicious actions. It is also in charge of managing authentication and encryption credentials and certificates. The client should be able to encode and encapsulate DLMS/COSEM traffic, as well as support the underlying media layers and communication profiles needed. The traffic generated by this component should be indistinguishable from traffic produced by a legitimate DLMS/COSEM client or server.

Currently there are certain communication profiles that are not supported by any open-source solution, but future developments and releases of these libraries would ultimately support their integration with this component. In later stages of development, proxy and man-in-the-middle attack capabilities could be added to this component for further research and testing purposes.

3.2.2 Sniffer

The sniffer component captures all the traffic from the environment being tested, and delivers it to the verification module of the tool, allowing a complete and in-depth analysis of the DLMS/COSEM messages. The sniffer gets a copy of the normal traffic as well as the testing messages inserted by the fuzzing module. Thus, through this component, ValiDLMS achieves the best coverage possible of the environment. Additionally, it can help in debugging tasks during the development of the fuzzing module.

3.3 Testing layer

The testing layer comprises components that implement *fuzzing* and *verification* modules. In the fuzzing module, a *fuzzer* component uses an *attack database* to run exploits and attacks, or otherwise induce the system to behave in an erroneous manner, while modifying packets with semi-random inputs and mutations. On the other hand, in the verification module, any traffic captured is dissected with two different parsers – *media dissector* and *DLMS dissector* – in resemblance with the DLMS/COSEM specification. The *verifier* component does most of the heavy lifting in the tool: it takes validation data from the *specification database* (Spec DB in the figure), and vulnerability data from the *vulnerability database* (Vuln DB in the figure), and checks: (1) if the DLMS implementation is in accordance with the specification; and (2) if there are exploitable vulnerabilities visible within and between the captured APDUs. If any deviation in specification or security violation is detected, the module outputs and logs a message, respectively, on the *display*

and *logger* components.

3.3.1 Fuzzing module

Fuzzing can be useful when assessing the security of a system, due to its ability to automate the testing with semi-random data. This allows a greater number of tests to be generated and performed in a relatively fast and thorough manner when compared with other methods. Random data is injected in order to check if it triggers some bug present in the system [27]. Therefore, fault discovery can be accomplished by monitoring the system for different forms of misbehaviour, for example, unexpected response messages [4]. It is also cheaper to employ than other alternatives, while also not requiring any access to the source code of the system being tested [19].

In general, fuzzers can be divided in three categories with respect to how they generate new packets or files. The first one is to feed completely random data as input to the target. This method is simple but is less likely to yield good results. In the second category, the mutation-based fuzzers alter pre-generated valid inputs in hope of discovering a vulnerability. In the third category, generation-based fuzzers create files or packets from scratch with the same intent based on a user specified model. Generation-based fuzzing enables higher coverage and a more comprehensive test suite when compared to mutation-based fuzzing [7].

The fuzzing module is composed by two components: *attack database* and *fuzzer*, which are detailed below.

3.3.1.1 Attack database

The attack database stores attacks and tests cases for use by the fuzzer. The users can define which messages and patterns are to be employed in order to induce errors and reveal bugs in the system (e.g., either the DTC or the meter). Specific exploits can be added as building blocks for larger and multi-stage attacks.

Entries in the attack database have a name for referencing and a small description, including limitations or caveats the attack depends on. The small description will assist the user to understand the test, hint at what the problem might be when the attack is successful, and also allow better security reports. For example, the attack sample in Fig. 3.2 shows both the definition of the attack, in order to be executable by the tool, and its description in natural language for easy comprehension and may also be used for interpreting results. The attack described in the example intercepts a message sent by a meter A, removes its authentication (resorting to the message authentication removal exploit, which would be defined in another entry), recovers the keystream and invocation vector, forges a message based on the tool's configuration, and then introduces the forged message in the network in place of the original message.

```

messageReplace(meterA)
{
  m1 < meterA.getCipheredMessage(predictedMsg)
  m1.removeAuthentication()
  k1 < recoverKeystream(m1,predictedMsgs)
  i1 < recoverIV(m1)
  m2 < forgeVulnerableMsg(m1,cgf)
  meterA.sendForgedMsg(m2,k1,i1)
}
{"Replaces a message with forged one using
the tool's global config. Cipher-only APDUs
need to be accepted and needs information
on predicted messages"}

```

Figure 3.2: Attack DB entry example.

3.3.1.2 Fuzzer

The fuzzer component reads the tests from the attack database and additionally may generate semi-random input for message fields, or deliberately generate and send incorrect messages. While the fuzzer executes attacks, the sniffer (from the interaction layer) captures the traffic to be analysed by the verifier component (see next section).

3.3.2 Verification Module

The verification module is intended to verify and analyse all traffic present in a DLMS/COSEM environment and can work independently without the fuzzing module, and it can also be used for monitoring real smart grid communications as it does not interfere with them at all. It receives captured traffic from the interaction layer's sniffer component, dissects it and abstracts the useful data from the lower layers, then verifies it using information loaded from databases containing both validation data and vulnerability data, and afterwards logs the results and presents them to the grid operator.

The verification module can also be used to aid the development of the fuzzing module, as it will allow the developer to verify if the exchange between the fuzzing module and the DLMS/COSEM environment is happening correctly and if the content in the messages is being generated as intended.

The verification module is formed by seven components: *media dissector*, *DLMS dissector*, *specification database*, *vulnerability database*, *verifier*, *logger*, and *display*, detailed below.

3.3.2.1 Media Dissector

The media dissector is in charge of dissecting the media layers within which DLMS packets are encapsulated. Ideally, this component supports as many of the four communication profiles described in the DLMS/COSEM specification as possible. Isolation allows independent development of this dissector layers. DLMS/COSEM currently supports four

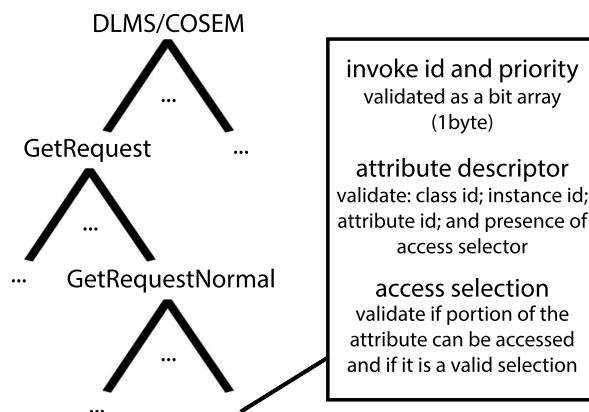


Figure 3.3: DLMS packet with content structured as a tree, for an example get request packet validation.

different communication profiles but it is expected that this number will increase as time progresses so it is only sensible to preserve that isolation in DLMS/COSEM related software design. Some transportation data, such as lower level addresses, may be passed upwards in order to verify more complex traffic patterns in the verifier component.

3.3.2.2 DLMS Dissector

The DLMS dissector receives preprocessed packets from the media dissector, parsing them in order to prepare DLMS packet fields and data to be inspected by the verifier component.

The best way to logically represent DLMS packets is in a tree diagram, as generic message types often contain more specific types, requiring branching out quite considerably. Therefore, ValiDLMS resorts to tree structures to keep information about a DLMS/COSEM packet organization and fields. For each packet, the dissector navigates in the tree and prepares the included data for inspection by the verifier component. Firstly, the dissector determines the type of the packet, and then notes down the specific context needed to interpret it correctly. Afterwards, at the leaf nodes of the tree structure, it prepares the required and optional generic fields. At this stage no inspection is executed by the verifier.

Fig. 3.3 shows an example tree structure for the navigation of a specific *get* request that can be used to access a multitude of data held by smart meters, for example, it can be used to retrieve the time attribute of a server's clock. Note that the tree on the left side of the image mainly represents the message's context which is necessary to interpret the data inside the rectangle on the right. The elements inside the rectangle on the right are generic, and can be contained in various different messages, for example, a *SetRequestNormal* message would also contain the *invoke id and priority*, *attribute descriptor*, and *access selection elements* in it but the different context would lead to a different interpretation.

3.3.2.3 Vulnerability Database

The vulnerability database will hold DLMS/COSEM vulnerability data. This component needs to be regularly updated in order to be able to detect the latest vulnerabilities and effectively help prevent vulnerability exploitation. Each entry in the vulnerability database has a set of mandatory components: a set of patterns and information which is used to detect the vulnerability in messages contents or in message patterns; a handle name or id for referencing; a date (or approximate date) of known disclosure; a risk assessment detailing how severe the vulnerability is, including a brief description of the possible impact its existence could have on the vulnerable system; and resolution advice on how to remove the vulnerability or where to look for implementation mistakes. Optionally, an entry may also figure vendor information if the vendors' implementation or equipment are vulnerable, and references which provide further information. The relevant set of data available in the database should be used in the log entries generated by the tool, resulting in a very concise and useful vulnerability report.

3.3.2.4 Specification Database

The specification database holds the data necessary to verify if the DLMS protocol is indeed well implemented in the sniffed packets, according to the DLMS standard. Information about data format, message syntax, and correct values should be present in this specification. Since structuring DLMS messages as tree diagrams is ideal, this specification should also be structured as a tree. This tree can then be compared against the data processed by the DLMS dissector, and parsed along the respective nodes to perform additional validation at the leave nodes with the correct context, such as value and data format validation.

3.3.2.5 Verifier

The verifier component is the most work-heavy component in the architecture. It has two main tasks: (1) it verifies that the DLMS/COSEM messages in comply with the protocol specification, and (2) it verifies if the received packets carry malcrafted data attempting to exploit some vulnerability. It uses complex pattern matching capabilities that allow it to verify message contents and patterns in packet sequences, evaluate tree traversal paths built from the captured and dissected packets, and verify them against the trees built using validation data from the specification database. The verifier receives the context from the type of messages identified by the DLMS dissector component, and the captured message buffer, and then builds a general context for each connection. Developers may extend this in order to be able to track individual equipment in a large DLMS/COSEM network by demanding additional information from the media dissector component. The protocol verification is done using the tree built from the specification database. The verifier finds

out whether or not existing fields should exist in a packet, if field data type is correct, if field values are within those defined in the specification for each field, and if the message patterns and order are correct, for example, check that requests precede responses, or check if the security level does not downgrade between sequential messages within the same connection.

When looking for vulnerabilities, the verifier needs to parse the data from the vulnerability database. Then, it is compared to the arriving packets to determine if there is a match. To improve performance, the information parsed from both databases needs to be consolidated at runtime, making the analysis of the packets more efficient, while still allowing both individual components to be maintained separately.

3.3.2.6 Display

The display provides feedback and online results to the user, employing colour schemes to facilitate the observation of found errors and vulnerabilities, and invalid messages, and distinguish between them quickly and easily. The information included in both database components is also utilized to provide comprehensive details about the findings. Colour schemes indicate the severity of the anomalies detected, showing by how much such anomalies put the network at risk or/and simply violate the DLMS/COSEM specification. Such colour schemes also help caught events stand out when using the tool is being used to actively monitor live traffic.

3.3.2.7 Logger

The logger produces readable and comprehensive logs from the running tests for offline inspection. The information presented in the log is intended to be even more complete than the one presented by the display, as it can be easily transmitted between security professionals assigned to different tasks, for example, the monitoring team could send it as a vulnerability report to the development team. Each log entry should contain a timestamp, details extracted from the databases about the problems found and information that allows finding the offending packet or packets. The included details will depend on whether a behaviour that violates the specification was detected, or a vulnerability was found.

Chapter 4

Proof of Concept

This chapter presents proof of concept for the framework proposed. First the testing environment is detailed, followed by a full description of our implementation of the ValiDLMS framework. Then, we present our evaluation method, the results, and our conclusions.

4.1 Testing Environment

A major industry partner from the electrical energy industry has kindly provided us access to a network and equipment emulating a small smart grid environment. The smart grid environment consisted in two data concentrators and six smart meters communicating using DLMS/COSEM over PRIME PLC links. The data concentrators acted as clients and were configured by the industry partner to carry out a number of predefined requests on the meters. They perform authentication with each meter using the LLS mechanism. However, no cryptography resources were in use in the tested configuration, as security was already offered and implemented by the PRIME communication layer.

4.2 ValiDLMS Implementation

This section presents how the ValiDLMS framework was implemented in its three layers. The limitations found in such implementation are detailed, accompanied by a description of the functionalities of the tool and their development. Note that the environment layer has already been described in the above section.

In the interaction layer, a hardware-based PLC sniffer was placed in a manner such that it can capture all packets being sent between the data concentrators and the smart meters. Captured packets were stored for further observation and analysis by the tool's verification module in the testing layer. The packets were saved in the well known PCAP format, which is the main capture file format used by most networking tools.

In the testing layer, the verification module was developed as an extension to Wireshark, a popular network packet analyser, taking advantage of its graphical interface and a few existing open-source DLMS dissectors programmed for it. The available dissectors had, however, to be enhanced, as we found some bugs that prevented the correct processing of the messages. The dissectors were fixed in order to be able to process segmented messages, and to fully process messages in spite of minor errors in packets, which is needed especially since the tool is expected to deal with erroneous messages sent by the fuzzer.

In developing the verification module, the verifier component functions were implemented as to perform checks on message buffers and using information loaded from the two database components. The current implementation performs a more thorough analysis of the *get* request messages, since they are the most prevalent in the DLMS/COSEM communications observed. The specification database was built using the COSEMpdu [5] schema, which has all DLMS/COSEM message syntax, and extending it with validation data relevant to the messages being analysed. The vulnerability database was built using known DLMS/COSEM vulnerabilities, as presented in Section 3.3.2.3. The display component was implemented using the wireshark API and settings to get the packet representation intended, using colours to mark packets with problems visibly. The logger component stored information about the findings in text files for later use.

The implementation of the fuzzing module was very limited, as we currently do not have access to an open-source library that implements communication in the PRIME PLC communication layer. For this reason, implementing the fuzzer component was beyond the bounds of possibility. However, traffic containing malcrafted data originating from attacks could still be captured by the verification module. The attack database was populated with invalid messages and tests related to the exploitation of DLMS/COSEM vulnerabilities. We could not inject our custom tests in our environment layer for aforementioned reasons, but some messages in pcap files were altered to simulate malcrafted packets and downgrade attacks. The attack database is strictly connected with vulnerability database, since the former contains the attacks defined to exploit the vulnerabilities described in the latter.

4.3 Experimental Evaluation

The objective of the experimental evaluation was to answer the following questions:

1. can ValiDLMS be used to validate DLMS/COSEM implementations effectively? (Section 4.3.1).
2. can ValiDLMS be employed to find vulnerabilities in DLMS/COSEM implementations? (Section 4.3.2).

3. can such a tool be utilized to test real industry based DLMS/COSEM implementations? (Section 4.3.3).

The ValiDLMS tool was evaluated using a set of PCAP files containing traces with packets captured from a DLMS/COSEM testbed with devices provided by our industrial partner. These devices and DLMS/COSEM implementation are being nowadays considered for deployment across certain regions of the country. Additionally, the altered messages referred in the previous section were also processed by the tool.

4.3.1 Validation

The verification module of the tool was used to analyse the PCAP files. As the included packets were not encrypted, all DLMS message fields were visible as plaintext, and so the traffic could be read in its entirety. The tool was able to correctly determine the type and syntax of messages, and could perform a deep and complete analysis of AARQ messages and *get* requests and responses. The *get* request messages were validated and were shown to be correct by our tool.

On the other hand, the tool found small inconsistencies in the AARQ messages. These messages seem to contain a bug, as an additional non-conforming byte is being included in them. This problem occurs after one of the fields of the AARQ header. In addition, the tool also discovered bugs in the processing of segmented messages. Although apparently not very complicated, all these problems can have an important impact if devices from multiple vendors need to interoperate in the same smart grid deployment, as they may cause malfunctions under certain conditions.

While processing the PCAP file, the tool displays for quick visualisation the erratic packets using colouring rules (see example in Fig. 4.1). Red corresponds to the most severe problems, meaning that the error can put at risk the security of the smart grid infrastructure. Light blue indicates messages that are in conformance with the protocol specification.

The fields of each packet can also be inspected in more detail, and problems are marked with colours. Once again, the colours are chosen depending on the severity of the errors. The tool also places informative labels near errors describing the problems found. Fig. 4.2 presents an example with the errors underlined in the package content.

The first question has positive answer, meaning that the implemented tool is capable of validating DLMS/COSEM implementations according to the protocol specification. Also, the tool had positive test results and was capable of withstanding incomplete or segmented packages, and tolerate and mark small one byte errors in packets.

184	nBox	DLMS	60	Bytes: AARQ Logical_Name low_level (abcd1234) cnf_blk: 0xffffffff
184	nBox	DLMS	60	Bytes: AARQ Logical_Name low_level (abcd1234) cnf_blk: 0xffffffff
167	nBox	DLMS	43	Bytes: AARE Logical_Name accepted cnf_blk: 0x001014 max-pdu-size:
167	nBox	DLMS	43	Bytes: AARE Logical_Name accepted cnf_blk: 0x001014 max-pdu-size:
137	nBox	DLMS	13	Bytes: GET-request Normal [Clock](time)
137	nBox	DLMS	13	Bytes: GET-request Normal [Clock](time)
142	nBox	DLMS	18	Bytes: GET-response Normal 2017/09/12 03:41:52 local (7_DST)
142	nBox	DLMS	18	Bytes: GET-response Normal 2017/09/12 03:41:52 local (7_DST)
155	nBox	DLMS	31	Bytes: GET-request Normal [Load profile with period 1 : Hourly :
155	nBox	DLMS	31	Bytes: GET-request Normal [Load profile with period 1 : Hourly :

Figure 4.1: Packets marked in wireshark using colour schemes.

No.	Time	Source	Destination	Protoc	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	184	nBox DLMS 60 Bytes: AARQ
2	0.001107	127.0.0.1	127.0.0.1	UDP	184	nBox DLMS 60 Bytes: AARQ
3	0.064474	127.0.0.1	127.0.0.1	UDP	167	nBox DLMS 43 Bytes: AARE

selected packet

packet DLMS/COSEM content

DLMS_COSEM 60 Bytes

APDU type : AARQ (0x60)

APDU src : Client

[Expert Info (Error/Protocol): unknown AARQ field : 0x80]

app_context : Logical_Name (1)

authentication : low_level (1)

passwd_CtoS : abcd1234

[Expert Info (Error/Security): Password in clear text: abcd1234]

user-information : be10040e01000000065f1f0400ffffff0400

dedicated-key? : no (0)

response-allowed? : no (0)

proposed-qos? : no (0)

proposed_dlms_ver : latest (6)

conformance block : 0xffffffff

client-max-pdu-size : 1024

Figure 4.2: Packet details with messages indicating problems found.

4.3.2 Security auditing

Since the weak LLS authentication mechanism was employed and communications were transmitted in the clear, it was not possible to test very sophisticated security problems (e.g., information leakage in ciphered packets). In any case, the tool found problems in the AARQ messages, since the password could be observed directly. This would let any DLMS/COSEM client connect using it, and have authenticated access to the meters.

Fig. 4.2 displays the selected package, which is marked due to a password discovered in the authentication process. We can confirm such security leak by visualising the DLMS/COSEM packet's content (bottom part of figure) and the label *Error/Security* near the error.

For each error found, the tool logs information (and alerts) in a log file. Fig. 4.3 shows an excerpt from a log file produced by the tool when it finds security leaks and non-conformance problems.

```
30-11-2017 16:25:01 Packet 183: Malformed packet: unknown AARQ field : 0x80
30-11-2017 16:25:01 Packet 183: Password in clear text in AARQ APDU. Password: abcd1234
30-11-2017 16:25:01 Packet 184: Malformed packet: unknown AARQ field : 0x80
30-11-2017 16:25:01 Packet 184: Password in clear text in AARQ APDU. Password: abcd1234
30-11-2017 16:25:01 Packet 191: Malformed packet: check for packet segmentation or incomplete message
30-11-2017 16:25:01 Packet 192: Malformed packet: check for packet segmentation or incomplete message
30-11-2017 16:25:01 Packet 203: Malformed packet: check for packet segmentation or incomplete message
30-11-2017 16:25:01 Packet 204: Malformed packet: check for packet segmentation or incomplete message
30-11-2017 16:25:01 Packet 213: Malformed packet: check for packet segmentation or incomplete message
30-11-2017 16:25:01 Packet 214: Malformed packet: check for packet segmentation or incomplete message
```

Figure 4.3: Log entries produced by the tool.

The second question is also answered positively, as the tool is able to detect security problems in DLMS/COSEM implementations.

4.3.3 Industry DLMS/COSEM implementations

The results show that the tool works in a smart grid environment and is able to validate and detect security flaws on an industry partner's DLMS/COSEM implementation, which allows us to answer positively to the third question. We would also note that being able to test a DLMS/COSEM network running on power-line communication is a great step forward as none of the existing tools support this communication profile. Theoretically, if the tool was running while directly connected with the network, online results could be produced. Unfortunately, the available equipment did not allow such experiment to take place in our setup.

Chapter 5

Conclusion

Smart grids have been taking the place of traditional electricity networks, since automatic meter reading was made possible through the modern grid infrastructure. New communication protocols have been developed, DLMS/COSEM being the most popular protocol discussed for employment in the smart grid and the one that offers the most functionalities, specially in terms of security mechanisms. Its impending deployment is commonly being discussed to be over power-line communication (PLC) links.

This dissertation focused on evaluating the security of the DLMS/COSEM protocol. After having studied the DLMS/COSEM protocol specification and its security state-of-the-art literature, the dissertation presented ValiDLMS, the first open-source framework for validating and securing DLMS/COSEM implementations in the growing smart grid industry, running over a power-line communication (PLC) profile, which is expected to be the norm for the industry in the recent future. The framework integrates a tool that was developed as an extension to Wireshark and was used to inspect an industry partner's DLMS/COSEM implementation. The results showed that ValiDLMS can effectively discover bugs, weaknesses and other non-conformance problems. We concluded that the framework can be applied by the industry to develop and integrate tools in order to test their own DLMS/COSEM solutions or those acquired from their business partners effectively. Further developments can extend tool to analyse live traffic, and detect anomalies in the grid's behaviour or provide online alerts of attacks being executed against the grid.

The current version our tool does not support all DLMS/COSEM message types, as only so much could be done in the scope of this dissertation. Completing the range of verifiable messages of our ValiDLMS tool is an essential task in order to assure the high coverage our framework is set to achieve. DLMS/COSEM has a great number of different messages and the main difficulty of this task lies in translating the complex specification requirements.

Developing an open-source library for communication on PLC networks is also an important task needed to assure that the fuzzing module in the framework works on such networks. It is a major limiting factor in the development of tools like the ones presented,

despite it being the physical communication medium most likely to be used in the smart grid in the future.

Bibliography

- [1] Distribution automation using distribution line carrier systems - Part 6: A-XDR encoding rule. Standard, International Organization for Standardization/International Electrotechnical Commission. IEC 61334-6:2000.
- [2] Information technology–ASN. 1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Standard, International Organization for Standardization/International Electrotechnical Commission. ISO/IEC 8825-1:2008.
- [3] Ross Anderson and Shailendra Fuloria. Smart meter security: a survey. *cl.cam.ac.uk*, 2011. available [online](#).
- [4] J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves. Vulnerability removal with attack injection. *IEEE Transactions on Software Engineering*, 36(3):357–370, 2010.
- [5] DLMS User Association. Cosempdu xml schema. available [online](#).
- [6] E Barker, D Johnson, and M Smid. Nist special publication 800-56a revision 2-recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. *Nist Special Publication*, New York, 2013.
- [7] Henrique Dantas. *Vulnerability Analysis of Smart Meters*. Master thesis, TU Delft, Delft University of Technology, 2014.
- [8] Klaas De Craemer and Geert Deconinck. Analysis of state-of-the-art smart metering communication standards. In *Proceedings of the 5th young researchers symposium*, 2010.
- [9] DLMS User Association. *COSEM Interface Classes and OBIS Object Identification System*, 12th edition, 2014. "Blue Book".
- [10] DLMS User Association. *DLMS/COSEM Architecture and Protocols*, 8th edition, 2014. "Green Book".

- [11] DLMS User Association. *Overview/Excerpts of the DLMS UA Coloured Books*, 2014. available [online](#).
- [12] Morris J Dworkin. *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC*. 2007.
- [13] Ahmad Faruqui, Ryan Hledik, Sam Newell, and Hannes Pfeifenberger. The power of 5 percent. *The Electricity Journal*, 20(8):68–77, 2007.
- [14] Stefan Feuerhahn, Michael Zillgith, Christof Wittwer, and Christian Wietfeld. Comparison of the communication protocols dlms/cosem, sml and iec 61850 for smart metering applications. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, pages 410–415. IEEE, 2011.
- [15] CEN CENELEC ETSI Smart Grid Coordination Group. Functional reference architecture for communications in smart metering systems. Technical report, CEN - CENELEC - ETSI, 2012.
- [16] CEN CENELEC ETSI Smart Grid Coordination Group. Smart grid information security. Technical report, CEN - CENELEC - ETSI, 2012.
- [17] CEN CENELEC ETSI Smart Grid Coordination Group. Smart grid reference architecture. Technical report, CEN - CENELEC - ETSI, 2012.
- [18] InGuardians. Advanced metering infrastructure attack methodology, 2009.
- [19] Willy Jimenez, Amel Mammar, and Ana Cavalli. Software vulnerabilities, prevention and detection methods: A review. In *in Proc. of the European Workshop on Security in Model Driven Architecture*, pages 6–13, 2009.
- [20] Cameron F Kerry. Digital signature standard (dss). 2013.
- [21] Phillip Laplante. Stakeholder analysis for smart grid systems. *Penn State*, 2010.
- [22] Netbeheer Nederland. *DSMR v4.0.4 P3 Companion Standard*, 2012. Dutch Smart Meter Requirements.
- [23] U.S. Department of Energy. *Smart Grid: An Introduction*. Office of Electricity Delivery and Energy Reliability, 2008. available [online](#).
- [24] Office of the National Coordinator for Smart Grid Interoperability. *Nist Framework and Roadmap for Smart Grid Interoperability Standards, Release 1.0*. 2009.
- [25] Dmitry Podkuiko. Vulnerabilities in advanced metering infrastructure. Master’s thesis, The Pennsylvania State University, 2012.

- [26] Marc Stevens, Pierre Karpman, and Thomas Peyrin. Freestart collision for full sha-1. Cryptology ePrint Archive, Report 2015/967, 2015. <http://eprint.iacr.org/2015/967>.
- [27] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 1st edition, 2007.
- [28] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 19–35. Springer, 2005.
- [29] Loren Weith. Dlms/cosem protocol security evaluation. Master’s thesis, Eindhoven University of Technology, 2014.