DLMS / COSEM protocol security evaluation

Weith, L.J.

Award date:
2014

Department of Mathematics and Computer Science
Information Security Technology Group

DLMS / COSEM Protocol Security Evaluation

Loren Weith

Supervisors:

dr. J.I. dev Hartog
Drs. Ing. D.H. Hut
dr R.H. Mak

Eindhoven, 18 March 2014

Contents

Acronyms

AA Application Association . . . 20
AAD Additional Authenticated Data . . . 13
AARE Application Association Response . . . 22
AARE Association Response . . . 22
AARQ Application Association Request . . . 21
AARQ Association Request . . . 21
ABRT Abort . . . 23
ACSE Association Control Service Element . . . 20
AES Advanced Encryption Standard . . . 12
AMI Advanced Metering Infrastructure . . . 3
ANSI American National Standards Institute . . . 9
AP Application Process . . . 20
APDU Application Protocol Data Unit . . . 19
APN Access Point Name
ASCII American Standard Code for Information Interchange . . . 35
ASN1 Abstract Syntax Notation #1 . . . 21
A-XDR Adapted eXtended Data Representation [32] . . . 24
BER Basic Encoding Rules [39] . . . 24
BSI Federal Office for Information Security . . . 7
CC Common Criteria for Information Technology Security Evaluation . . . 7
CENELEC CENELEC . . . 5
CLS Controllable Local System . . . 5
COSEM Companion Specification for Energy Metering . . . 5
DER Distributed Electrical Resources . . . 2
DLMS Device Language Message Specification . . . 5
DSMR Dutch Smart Metering Requirements . . . 4
DSO Distribution System Operator . . . 2

EAL Evaluation Assurance Level . . . 8
EU European Union . . . 4
ERP Enterprise Resource Planning . . . 3
FTDI Future Technology Devices Internations, Ltd .[2] . . . 34
GCM Galois Counter Mode . . . 9
GHASH GCM Hash . . . 15
GMAC Galois Message Authentication Code . . . 13
GO Grid Operator . . . 4
GPRS General Packet Radio System . . . 6
HDLC High Level Data-Link Control . . . 6
HE Head End System . . . 3
HLS High Level Security . . . 22
ICV Integrity Check Value . . . 13
IEC International Electrotechnical Commission . . . 25
IP Internet Protocol . . . 6
IT Information Technology . . . 7
ITSEF Information Technology Security Evaluation Facility . . . 8
ITU International Telecommunications Union . . . 20
ITU-T ITU Telecommunication Standardization Sector . . . 20
IV Initialization Vector . . . 13
jDLMS jDLMS [5] . . . 24
LLS Low Level Security . . . 21
MAC Message Authentication Code . . . 12
MD5 Message Digest 5 . . . 22
MID Measuring Instruments Directive 2004/22/EC . . . 4
MITM Man in The Middle . . . 32
MON Remote Monitoring . . . 31
NIST U.S. National Institute for Standards and Technology . . . 8
NSCIB Netherlands Scheme for Certification in the Area of IT Security . . . 7
OBIS Object Identification System . . . 24
OID Object Identifier . . . 21
PLC Programmable Logic Controller . . . 3
PP Protection Profile . . . 7
RFC Request for Comments . . . 57
RLRE Release Response . . . 23

RLRQ Release Request 23
SCADA Supervisory Control and Data Acquisition 3
SGAM Smart Grid Architecture Model 1
SGCM Sophie Germain Counter Mode 18
SHA1 Secure Hash Algorithm 1 22
SIM Subscriber Identity Module 6
TCP Transmission Control Protocol 19
TLS Transport Layer Security 57
TMTO Time / Memory Trade-Off 44
TRM Interactive Terminal 31
UART Universal Asynchronous Receiver / Transmitter 34
UDP User Datagram Protocol 19
USB Universal Serial Bus 33

List of Figures

List of Tables

Chapter 1

Introduction

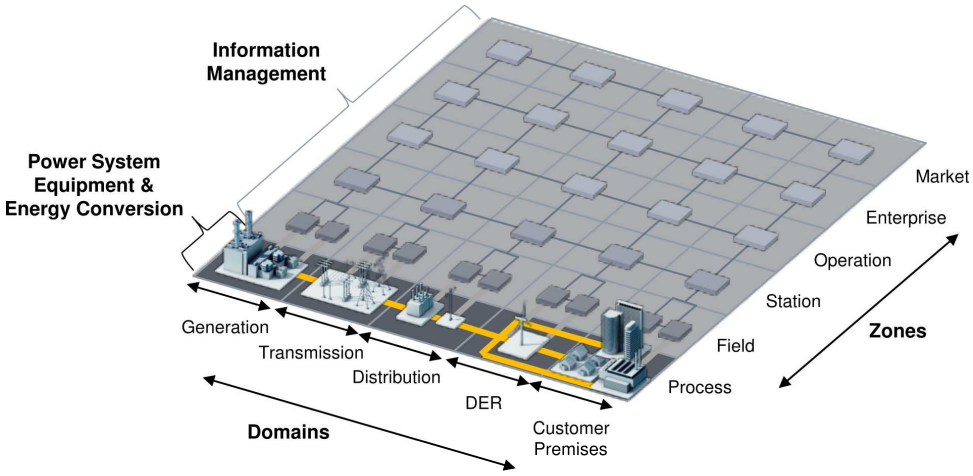

Figure 1.1: SGAM Smart Grid Plane (Reproduced From [15])

Before directly tackling the main topic of evaluating the security of smart meter communication protocols, let us first start by reviewing the larger context of a smart grid. The European Commission Task Force for Smart Grids define a "Smart Grid,"[28] which we refer to without capitalization in the sequel, as "an electricity network that can cost efficiently integrate the behavior 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."

1.1 Smart Grid Architecture Model

The Smart Grid Architecture Model (SGAM)[15] defines a reference model with which to discuss the hierarchies of power system equipment (domains) and the information management systems (zones) that control them. The two dimensions are combined to produce a convenient spacial representation of the relationships between control systems and the processes that they control, known as the SGAM Smart Grid Plane. The third

layer represents layers of interoperability and describes relationships between components at multiple layers. Our use of the model is limited to the communication level so we neglect the others, whose can be found in [15].

1.1.1 Domains

The SGAM[15] categorize the energy conversion chain in the following domains whose descriptions are paraphrased from a similar table in [15]:

Bulk Generation	The bulk generation domain includes power plants that generate power on a large scale.
Transmission	The transmission domain is concerned with long distance transport of bulk electricity.
Distribution	The distribution domain is where bulk power is distributed over a more localized geography to customers.
DER	Distributed Electrical Resources (DER) are contained within this zone and are generation facilities with outputs from 3kW to 10,000 kW and are controlled directly by a Distribution System Operator (DSO).
Customer Premises	Customer premises can range from homes to large scale manufacturing operations and any one of them may may also provide generation services, though generally at a smaller scale than the facilities in the DER or Bulk Generation domains[15].

1.1.2 Zones

Zones map approximately to the *Purdue Reference Architecture*[62], which is a reference model for enterprise architecture. The following are descriptions of the SGAM zones, also paraphrased from [15]:

Market	The Market zone is the zone where systems outside the scope of a single enterprise exist. When a system within an enterprise interacts with e.g. other companies, regulators, industry groups, etc., the systems are considered to operate in the Market zone.
Enterprise	Systems in the Enterprise zone are concerned with management functions that effect entire companies at a business level and that may cross multiple domains. Examples of systems at this level would be Enterprise Resource Planning (ERP) or billing systems.
Operation	The operational level systems are concerned with management of systems that compose entire domains, such as field operations. Systems in the Operation zone include Advanced Metering Infrastructure (AMI) Head End Systems (HEs), Supervisory Control and Data Acquisition (SCADA) systems that monitor and control various aspects of the energy conversion chain.
Station	Systems at the station level are control systems that operate on field systems that manage / monitor actual physical processes in a particular domain but with a more limited geography than e.g. an AMI Head End System or SCADA system. A station level system could be a neighborhood level data aggregator for metering data collection, for example.
Field	Field level systems directly monitor and control physical processes taking place in the energy conversion chain. An example of such a system could include a Programmable Logic Controller (PLC) regulating the amount of fuel supplied to a generator such that it produces as close to a 50Hz waveform as possible or a smart meter at a customer premise connecting or disconnecting power.
Process	A process is simply a physical process in the energy conversion chain. It could be anything from a fission reaction to the ensuring that the blades of a wind turbine are not moving at an unsafe rate of speed.

1.2 Smart Meters and AMI

Smart meters function as part of an overall AMI that can operate, at a general level, in the field, station, and operational zones of the customer premise domain. We use the AMI reference model[14] to clarify what exactly is meant by smart meters and the AMI that it is part of.

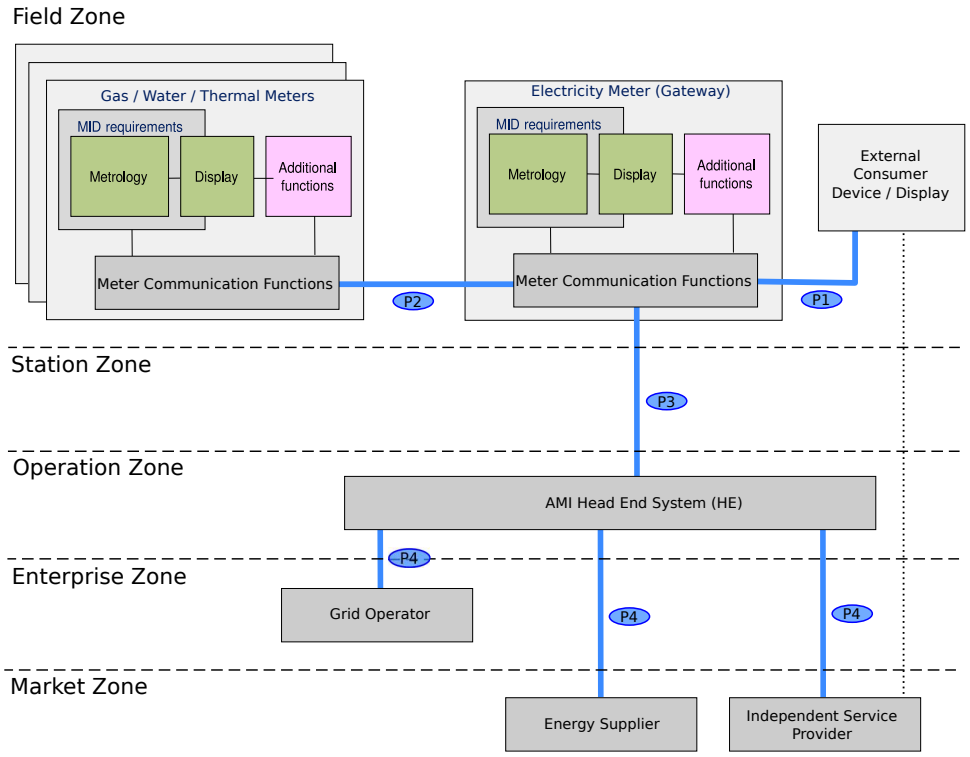

Figure 1.2: DSMR Components in SGAM Zone Context (Adapted from [14])

The Measuring Instruments Directive 2004/22/EC (MID)[49] and [14] define a *meter* as an "instrument for measuring, memorizing and displaying the consumption of a commodity" and a *smart meter* as a "meter with additional functionalities, one of which is data communication." While the definition of a smart meter is general and some results are likely applicable to multiple types of smart meters, we consider only electricity meters in the context of the Dutch smart grid.

The center of the architecture is the AMI HE, which doe all of the communication with smart meters. The AMI HE performs the direct communication and control work and then feeds that data to internal customers within their own company and with upstream energy suppliers and service providers who need access to carry out work on behalf of other parties in the system. The zones help to differentiate between what happens inside an organization operating an AMI and outside.

1.3 Dutch Smart Metering Requirements

The Dutch NTA-8130[48] defines the technical requirements for electricity, gas, and thermal energy meters for the Netherlands, similarly to how [49] defined them at the level of the European Union (EU) and the Dutch Smart Metering Requirements (DSMR)[27] is a companion specification from NetBeheer Nederland, a consortium of Dutch Grid Operators (GOs) that provides more specific requirements that meet the needs of stakeholders in the Netherlands.

The DSMR specifies a set of ports that must be present on any smart meter deployed in the Netherlands in the context of the overall smart grid, together with specific functional and security requirements for each [1] broken out into the companion standards[24] [25] [26] [23]. Every DSMR compliant smart meter is required to support several interfaces with accompanying security and functional requirements. A brief description of each DSMR port as well as the P4 port used to communicate with enterprise and market level systems in included in table 1.1. The communication relationship of systems operating in the context of the Dutch smart grid (fig. 1.2) is presented in terms of the CENELEC (CENELEC) *Functional Reference Architecture for Communications in Smart Metering Systems*[14], which we refer to in the sequel as the AMI reference architecture.

Since our work is related only to the Device Language Message Specification (DLMS) / Companion Specification for Energy Metering (COSEM) protocol, we are only interested in ports P0 and P3 but we include them all to provide context.

1.4 Security Implications of Smart Meters

Smart grid technologies are facilitating for example, the widespread deployment of renewable energy production capabilities directly to consumers, e-mobility, where electric vehicles move from place to place, charging wherever they happen to park, and Controllable Local Systems (CLSs) like air conditioners or refrigerators that can adjust their energy consumption based on energy costs and better match consumption to production[44]. With the smart grid, production and consumption can both be manipulated for energy savings, rather than just adjusting generation to match consumption[44]. Smart meters are the component at the consumer level that makes dynamic tariffs possible, which in turn inform CLSs so that they can reduce their power consumption when the grid is under high load.[44]

On the other side of the coin from the environmental and cost advantages, smart meters have security implications including invasion of privacy where power usage can leak information about daily routine, work schedule, to facilitate e.g. burglary, to nightmare scenarios where nations are crippled after having their power turned off in a cyber attack[18].

1.5 Research Question

The question to be answered by this thesis is the following: (a) Determine a repeatable method (b) to evaluate the security of smart meters (c) with respect to their ability to protect stakeholder interests, (d) specific to the DLMS/COSEM protocol as deployed in the Netherlands, and (e) practically implement it. Our primary focus will be upon parts d and e, which are independent of the answers to the other parts of the research question and can be addressed without the requirement of widespread stakeholder participation. Our objective and approach along these lines is set forth in section 1.9.

[1] P0 is not included, presumably considered to be vendor-specific

P0	This port, present only on electricity meters, is used for communication with external devices used at the times of installation and maintenance. The main DSMR document makes no mention of a companion standard describing the details of the P0 interface. That said, it is also not uncommon for a P0 port to conform to IEC-62056-21[33]. In the case of the meters tested in the lab, they supported DLMS / COSEM over High Level Data-Link Control (HDLC) (IEC-62056-21 protocol mode E.)
P1	The P1 port is a read-only interface for communication between metering installation and auxiliary equipment. DSMR[24] defines a standardized RJ-11 serial interface with a "request" input line to request data from the meter, a data output line to receive it from and a 5v, 100 mA power supply to power devices connected to the port.
P2	The P2 port is for communication with other metering equipment within a household. DSMR specifies that the electricity meter at a customer premise provides the communication facilities for e.g. water and gas meters at the same location.
P3	The P3 port is used to communicate with the head-end of an AMI. The DSMR prescribes that communication on this port support Internet Protocol (IP) over General Packet Radio System (GPRS). Naturally, "the Telecom Providers must offer a dedicated APN per Grid Operator to which those grid operators smart metering Subscriber Identity Module (SIM) cards have exclusive access"[23].
P4	The P4 port is the port used to communicate data between the Head End System (HE) at the operational level, internal business operations within the GO, and external stakeholders including energy suppliers and independent service providers[27].

Table 1.1: DSMR Communication Port Identifiers

Items a and b already have well established and practical solutions (section 1.6.) However item c, which is a very large question requiring consensus from a diverse group of stakeholders is not so straight-forward. They must agree on what their security requirements are, how they must be protected, and how certain they need to be that the protections are enough. While the specific question has not yet been answered, the process for creation of a Common Criteria for Information Technology Security Evaluation (CC) Protection Profile (PP) is applicable has already resulted in an official protection profile in Germany. Section 1.7 goes into more depth on the subject of PPs.

1.6 Repeatable Method for Evaluation

The CC provides a widely accepted framework for evaluating whether Information Technology (IT) systems meet security criteria set forth by user communities and for different user communities (schemes) to agree upon the equivalence of their requirements so that, for example, a firewall meeting the Dutch requirements for security at a particular assurance level would also be acceptable under other protection profiles that have been found to be equivalent, without the developer needing to re-certify for every market in which they wish to participate.

CC is widely accepted and provides a ready-made ecosystem of schemes like Netherlands Scheme for Certification in the Area of IT Security (NSCIB), that develop and maintain security standards called PPs for various types of devices, evaluation facilities that are prepared to carry out evaluations based on the same protection profiles. Due to those advantages alone, CC is a sensible high level solution to requirements (a and b.)

1.7 Stakeholder Requirements

Before a determination can be made as to whether stakeholder security requirements are met by an IT system, it must first be determined what their needs are, which is certainly among the largest (albeit implicit) questions before us, and one whose scope greatly exceeds that of a masters thesis,

In the context of CC, a protection profile codifies these basic security requirements and as an example, the German Federal Office for Information Security (BSI) has already created a protection profile for what they call a "Smart Metering Gateway,"[44] that encapsulates the security functionality of existing smart meters and also serves to enforce privacy and transparency as to the activities of e.g. grid operators with respect to the meter. The separation was presumably done to encapsulate security functionality so that it is possible to safely ignore the details of the meter as a whole.

The fact that Germany has explicitly chosen common criteria and the United States Department of Energy has at least shown deference to the methodology in its "Advanced Metering Infrastructure Security Requirements" technical report [20] that, while not a formal protection profile, follows the format closely and acknowledges its influence. The precedent set by Germany, and to a lesser extent by the US on the appropriateness of

CC for that purpose, combined with wide acceptance for the Common Criteria in other security-critical devices like firewalls, chip cards, operating systems, etc. provides further evidence to support CC as an appropriate direction for the Netherlands.

The NSCIB has not yet promulgated a formal Protection Profile for smart meters, but NetBeheer Nederland, a consortium of Dutch electrical grid operators has published its DSMR specification, which serves as the current set of security (and general) requirements for smart meters used in the Netherlands. Clearly, the standard itself is helpful but more helpful to the cause of ensuring security is NetBeheer itself, which is in a position to encourage the formation of a committee of stakeholders to develop a formal Protection Profile and ultimately transform DSMR into a purely functional specification, allowing the PP committee to handle stakeholder security requirements and determine required Evaluation Assurance Levels (EALs) required for acceptance in the Netherlands.

Assuming a well executed PP creation process, its assurances will be sufficient to meet the requirements of the majority of stakeholders, developers will need to have their products (at most) evaluated by an Information Technology Security Evaluation Facility (ITSEF) under each scheme for each user community, though it is likely that the schemes will negotiate equivalence relationships or developers will have their products evaluated against multiple security targets. The end result is that the current situation where each asset owner is responsible for their own security evaluations, that function will be a standardized service that is paid for by each developer, rather than each customer.

1.8 Related Work

Very little has been said specifically about the DLMS / COSEM security protocols and among the most detailed evaluations are [42], which approaches the problem at a theoretical level, breaking out classes of attacks and their likelihood and some students from the University of Colorado[45] have taken a similar approach, but at a higher level of abstraction.

In terms of evaluation, the Common Criteria for Information Technology Security Evaluation framework seems the likely choice at a regulatory level, considering that it is already widely used within the military and enterprise spaces for general purpose computing and networking and has been favorably referenced in publications from U.S. National Institute for Standards and Technology (NIST) and is the basis of the new German smart metering gateway protection profile[44] (section 1.6.)

Among the the most complete attack methodologies encountered was [40], which covers an AMI from enterprise to field in the customer premise domain. While the methodology does include some discussion of cryptographic issues, there is a clear bias toward identifying and exploiting hardware and software issues specific to a particular device. Cryptographic tests are discussed, but only tests of basic components like random number generation and searches for key material in firmware dumps. [13] takes a similar approach and reviews some of the security failures that have been uncovered by the C4 security company in Tel Aviv had results at a similar scope – a list of vulnerabilities

are enumerated that that include authentication bypass, buffer overflow due to incorrect parsing, etc. They attacked software implementations, for the most part ignoring the protocol.

The very closest related work, and the work that inspired the Taco shell (section 6.9,) was the Termineter project[10] that created an interactive shell (also based on the Python cmd module) for attacking the American National Standards Institute (ANSI) C12.19 protocol that is most commonly deployed in the United States.

Apart from very high level work like the examples above, we were unable to locate any documentation of vulnerabilities for the DLMS / COSEM protocol. In fact, finding any public information at all about the security protocol and particularly the cryptography posed a significant challenge.

1.9 Objective & Approach

As stated above we need not solve the problem of a repeatable high level methodology (section 1.6) and cannot hope to produce an adequate protection profile, though there is precedent supporting the approach (section 1.7.) Even if the Netherlands does not adopt the CC framework, we can reasonably assume that some framework will be chosen and that vulnerability testing will be a part of the process.

For that reason, the objective of this thesis, is to present a vulnerability evaluation framework to identify protocol and implementation level vulnerabilities and test for them. The practical implementation covers the protocol specifics and implementation requirements of the Netherlands as defined in DSMR.

We approach the objective by reviewing the existing standards in sections 1.6, 1.7, and 1.8; giving a vulnerability testing methodology complementary to them (chapter 2;) and a practical implementation. Our implementation includes a discussion of the Galois Counter Mode (GCM) authenticated streaming mode (chapter 3,) an overview and analysis of DLMS/COSEM (chapter 4,) a set of tools to facilitate evaluation and testing (chapter 6,) and a report of discovered vulnerabilities (chapters 5 and 7.)

Chapter 2

Testing Methodology

The methodology that drove our practical work differs from [40] in that rather than producing results applicable to a particular piece of hardware, its outcome is a set of generic, re-usable test cases that can be used with most meters adhering to the standard under evaluation – in our case DLMS / COSEM.

Since there is not yet an official security standard, asset owners and device manufacturers are are responsible for their own testing to provide confidence that the devices they build / operate are secure. The attack methodology in [40] is likely to mostly produce device-specific vulnerabilities with a few generic ones and ours is likely to produce more or less all generic vulnerabilities, missing most device-specific issues.

The approach presented here is intended to be complementary to the exhaustive hardware evaluation described in [40]. From our evaluation methodology for protocols, comes regular and clear documentation of the protocols, their potential shortcomings, and exploits that can be used to test meters from multiple vendors. The attack methodology in [40] can be carried out against devices that have been passed the generic protocol-level attacks that can be run against any implementation.

2.1 Process

In this section, we present an iterative methodology and laboratory architecture for generating vulnerabilities, testing criteria to determine if they are present, and developing exploits that operate across vendors supporting the same protocols. This process is an iterative one – the information learned from the last iteration is then combined with further research to produce output that provides a further refined understanding of the protocols and their vulnerabilities.

2.1.1 Protocol Evaluation

Understand basic protocol, the component technologies of the protocol and their purposes. If no documentation exists for the protocol, such documentation should come from this process. The understanding drives an evaluation of the protocol of the protocol with respect to the required environment, noting protocol features that might be useful in an attack on the overall system, but that may not be exploitable by themselves. The list should be based on commonly accepted protocol security requirements like those listed in [51] and in particular [40].

2.1.2 Cryptography

Understand the details of cryptographic components, together with their requirements, strengths, and weaknesses.

2.1.3 Vulnerability Enumeration

From the list of potential vulnerabilities in chapter 7, develop test procedures to verify whether any of the vulnerabilities are present and automate the testing where practical.

2.1.4 Generate Exploits

Develop exploits that make use of one or more vulnerabilities in chapter 7 to be presented to system developers or asset owners as evidence of exploitability and potential impact. Having an exploit is irrefutable and immediate proof that an issue exists and that it results in the outcome stated – it is easier to convince someone to act on a demonstration than on a theory, no matter how well founded the theory is.

2.1.5 Tooling

Develop tools for observation and decoding of legitimate traffic, manual and automated manipulation of legitimate traffic to nefarious ends, as well as libraries and other ancillary tools to facilitate exploit development.

2.2 Validation

The process is validated in the chapters to follow: chapter 4 is the outcome of steps 2.1.1 and 2.1.2. The tools developed as called for in section 2.1.5 are described in chapter 6, and chapter 7 covers the the vulnerabilities and exploits.

Chapter 3

Galois Counter Mode

When evaluating the security of a system, it is important to assume as little as possible and look at the details of its cryptography, as noted in chapter 2, section 2.1.2. To that end, we consider the details of Galois Counter Mode (GCM). GCM is an authenticating stream cipher that performs its encryption in a manner very similar to that used in counter mode[30] (section 3.3) but with the addition of a high performance message authentication code to compensate for the ease of performing known plaintext attacks against counter mode. The name Galois Counter Mode is a reference to the main building block of it's Message Authentication Code (MAC), the Galois Field $GF(2^{128})$.

3.1 Features

The following are some of the main features of the streaming mode:

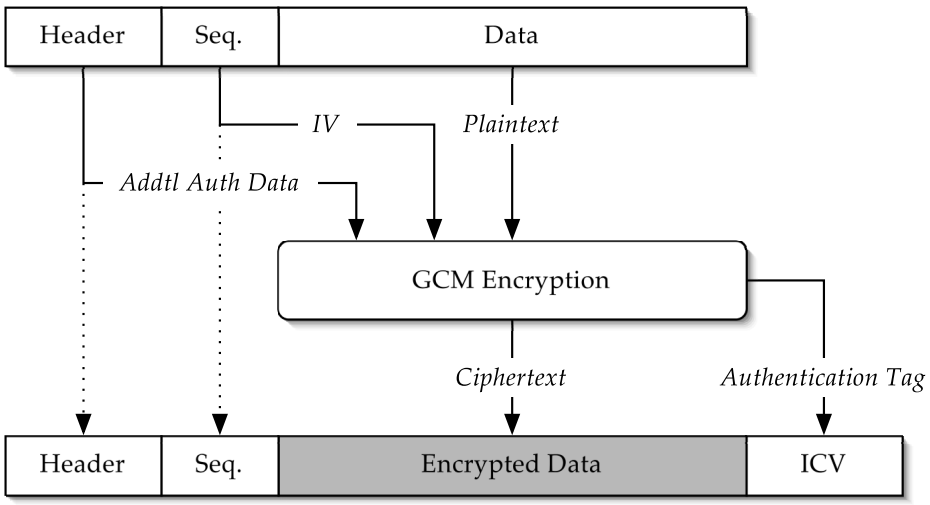

Figure 3.1: GCM Overview (Reproduced from [47])

a) The gate count in hardware implementations was reduced by using the Advanced

Encryption Standard (AES) encrypt operation to generate a keystream that is then xor'd with the plaintext, eliminating the need for the decrypt operation and the gates required to implement it. Naturally, the simplification can also result in reduced code size in software-only implementations.

b) Many of the functions performed in the authentication and encryption operations can be computed out of order, facilitating the development of highly parallel hardware implementations.

c) Additional Authenticated Data (AAD) support provides a convenient way to tie related plaintext data like network headers to an encrypted payload without additional overhead. GCM also supports authentication of purely plaintext messages called Galois Message Authentication Code (GMAC). In GMAC, plaintext data is processed as AAD but no plaintext is encrypted with it, resulting in only an authentication tag.

d) The Integrity Check Value (ICV) is computed in such a way as to provide an *Incremental MAC*[19]. The incremental MAC feature is helpful because someone in possession of the AES encryption key used for ciphering can make changes to a message whose ICV has already been computed and then update the ICV to reflect the change instead of needing to re-create the ICV from scratch.

3.2 Keystream Generation

GCM is a ciphering mode of operation where confidentiality is achieved by using a block cipher (128-bit AES in most cases) to generate blocks of pseudorandom bits based on the encryption key and a unique Initialization Vector (IV). Uniqueness is important because repetition of the keystream in multiple messages can result in very serious consequences including revelation of plaintext and loss of message integrity protection[41].

The GCM keystream

$$k = K_0, K_1, ..., K_n \tag{3.1}$$

is a function of a public IV and a shared, secret encryption key. Per the specification, the IV can be any length, but an IV that is not exactly 96 bits long is considered insecure because of the weak manner in which such IVs are transformed into counters for keystream generation (section 3.5). The insecure IVs are discussed briefly in section 3.5 but excluded from this section because it is not relevant to the DLMS/COSEM protocol. For the full details, please refer to [59].

Each 128-bit block of keystream is computed by using the chosen block cipher, 128-bit AES, the public IV, and a block counter, expressed as a 32-bit big-endian unsigned integer that starts with the value 1. For block n of the keystream, keystream block K_n = `EncryptAES( IV || Counter )`. Block zero is reserved for encrypting the ICV (section 3.4) and the remaining blocks are used to encrypt payload. If the number of bytes in a message is not divisible by the block size of the cipher, which for our purposes, will be assumed to be 128-bit AES unless otherwise stated, then the last message block

is padded to the size of one block and the output is truncated appropriately to facilitate application of GMAC (section 3.4.)

3.3 Encryption / Decryption

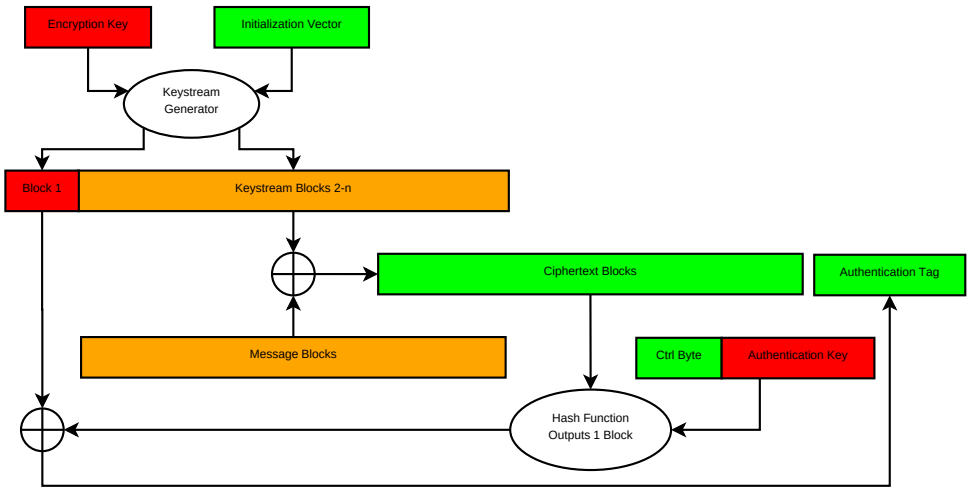

Figure 3.2: GCM Encryption Operation

Recall that for a bit a, $a \oplus a = 0$ and $a \oplus 0 = a$. When we write $a \oplus b = c$, where a, b, c are bit strings, it means that each bit in c is the outcome of the XOR operation on the corresponding bits in a and b. Unless otherwise specified, variables are bit strings.

The encryption of plaintext p using keystream k to produce ciphertext c is performed by computing $k \oplus p = c$ as illustrated in fig. 3.2.

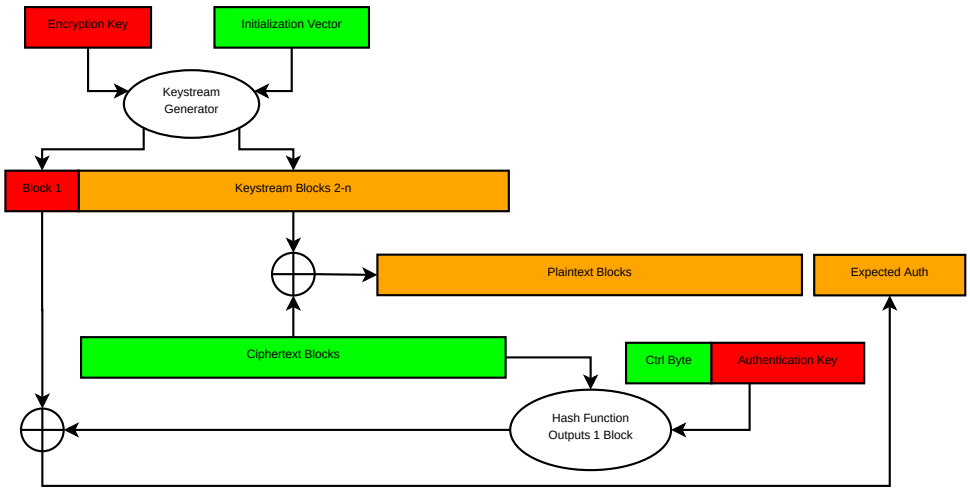

Figure 3.3: GCM Decryption Operation

Since the ciphertext c is the information transmitted between client and server, both an eavesdropper and legitimate participants have the information. Given c and either p or k from 3.1, then we can compute the third piece of information by either performing the normal decryption operation and computing $p = c \oplus k$ or if we happen to know the

plaintext already, we can retrieve the keystream k by computing $k = c \oplus p$. Once the keystream k is recovered, it can be re-used to validly encrypt another value that is the same length or shorter. If an application uses GCM but ignores the ICV then such a forgery (section 7.3.3) will not be detectable.

3.4 Message Authentication

Here, we present the MAC algorithm used in GCM in the same simplified form as in [55] in order to provide the proper context for the known weaknesses in section 3.5.

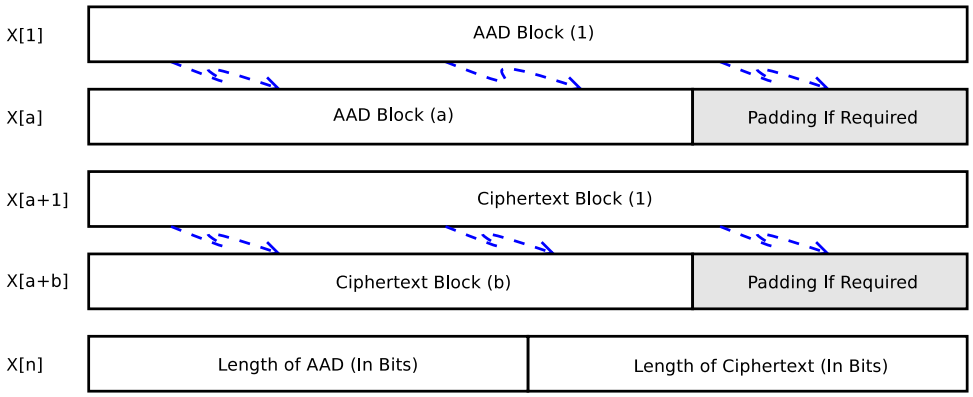

Figure 3.4: GMAC Block Layout

The description starts after all the 128-bit ciphertext and AAD blocks have been collected together as a list of 128-bit strings organized as illustrated in fig. 3.4. Zero bits are added to the end of the last block of AAD or ciphertext in order that they fill out an entire 128-bit block. The lengths in X_n are 64 bit integers with their most significant bits first.

Let a be the number of AAD blocks, b be the number of ciphertext blocks, and there is one remaining termination block at the end. Let $n = a + b + 1$ be the total number of AAD, ciphertext, and termination (counter) blocks.

The GCM Hash (GHASH) algorithm itself acts upon elements of $GF(2^{128})$ whose elements are polynomials of degree 128 modulo $1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$ with coefficients in $\mathbb{Z}_2$. We consider each X_i to be an element of the same field. Naturally, the addition operation is the same as the exclusive or operation because addition modulo 2 has the same truth table as the XOR operation. Let us further define H `= EncryptAES( Key,` 0^{128} `)` where H is also interpreted as a finite field element. We compute Y as:

$$Y = K_0 + \sum_{i=1}^{n} X_i \cdot H^{n-i+1} \tag{3.2}$$

since finite fields are commutative, order of addition does not matter.

3.4.1 Incremental MAC

The incremental MAC functionality follows from the fact that the GHASH uses only addition and multiplication over a finite field. Since finite field operations are commutative and inevitable, the contribution of an existing block can be removed without re-computing the rest of the blocks and its replacement or an additional block can be added to the hash with less computation than re-computing the entire hash over again.

3.5 Known Weaknesses

Since GCM is basically the same as counter mode, apart from it's authentication mechanism, it is not surprising that of criticism of GCM is is related to its message authentication since the shortcomings of counter mode are well established.

Before proceeding, let us clarify that it is difficult to directly apply any of the generic results of attacks on standard GCM to DLMS / COSEM because they can all safely assume that the AAD is known to an attacker and that assumption does not hold for GCM. Until more research is done to determine which attacks have variants that apply to DLMS / COSEM. A good place to start might be the forbidden attack of Joux[41], that results in exposure of the GCM MAC secret H upon repetition of an IV.

The vulnerabilities discussed by Ferguson in [29] are the following:

a) The strength of an authentication tag of n bits only provides $n - k$ bits of security, where the message length is 2^n[29]. Every time a forgery is found, the process of finding the next forgery is easier and information about the hash key H is leaked. When all bits of H are leaked, forgeries will always be possible[29].

b) The NIST version of the specification does not mention the importance of never using multiple authentication tag lengths with the same key[29].

c) If an IV other than 96 bits long (in which case no collision is possible,) then GCM will be expected to re-use a counter after on average, 2^{64} message blocks have been encrypted. The reason for the problem is that the GHASH algorithm is run against the unusual length IV, forming what is essentially a pseudorandom number generator and that generator will eventually repeat an output.

   In the case where there is no AAD and the attacker has full knowledge of the plaintext of the messages, the collision will result in recovery of H (see section 3.4) and complete loss of the ability to detect forged messages[29].

Joux explains in [41] that the original submission from McGrew and Viega was altered by NIST in such a way as to increase the impact of using an IV that is more than 96 bits long.

3.5.1 Weak Keys in GCM

Saarinen presents[55] a weakness in GCM based on the algebraic setting in which GHASH is computed. When looking at GHASH presented as it is in equation eq. (3.2), the nature of GCM's protection against re-ordering of blocks becomes plain. Since the summation can be computed in any order, if $H^i = H^j, i \neq j$, then the positions of the corresponding ciphertext blocks can be swapped. In order for the hash to return a different value for all orderings of blocks then the order of the element H ($|\langle H \rangle|$) must be greater than the number of blocks hashed by GHASH. If $c = |\langle H \rangle| < n$ then blocks B_a and B_b where $a \equiv b \mod c$ can be swapped without effecting the hash[52].

The weakness comes from McGrew and Viega's choice of $GF(2^{128})$ for use in the GHASH algorithm. The order of its multiplicative group is $2^{128} - 1 = 3 \cdot 5 \cdot 17 \cdot 257 \cdot 641 \cdot 65537 \cdot 274177 \cdot 6700417 \cdot 67280421310721$, resulting in $2^9 = 512$ divisors, many of which are quite small, resulting in the existence of weak values of H with short cycles and thus, weak keys that result in weak values of H.

Attacks are clearly possible for certain values of H, so the next natural question is whether an attack is likely to be effective against GCM in the context of DLMS / COSEM. The ACSE session establishment protocol described in (section 4.2.3) is used, among other things, to exchange 16-bit values that indicate the total size of protocol messages that will be accepted by both client and server. Given that the messages sizes are only 16 bits wide, the maximum possible value is 65535 and since we need only a reasonable upper bound on maximum block count, let us assume that there is no overhead and that 65535 byte ciphertexts can be transferred as 4096 128-bit blocks, even though the actual number of blocks will be smaller than that due to overhead.

Given that assumption, the MAC will be computed over those 4096 blocks, 2 blocks of AAD, and one block containing the lengths of AAD and ciphertext, making a total of 4099 blocks. In order for DLMS / COSEM's use of GCM to be vulnerable to the attacks described in [55], a device must be using an AES key that results in a hash key $H \in GF(2^{128})$ with order less than 4099.

Since we know that for each divisor of a cyclic group, there exists exactly one subgroup of that order, we can list all of the subgroups with orders less than 4099, then use the Euler totient function to determine the number of elements that generate those groups to avoid counting elements present in multiple subgroups multiple times. First, we enumerate the divisors of the order of the multiplicative group of $GF(2^{128})$ whose values are less than 4099, which are: 1, 3, 5, 15, 17, 51, 85, 255, 257, 641, 771, 1285, 1923, 3205, 3855. For each of those divisors, we use the Euler totient function to compute the number of generators in each subgroup: 1, 2, 4, 8, 16, 32, 64, 128, 256, 640, 512, 1024, 1280, 2560, 2048, which we add up to arrive at the total number of elements in $GF(2^{128})$ that generate subgroups of order less than 4099 – there are 8575 of them.

Assuming that there is somewhere close to a 1:1 mapping of AES keys and values of the hash key H derived from them and the use of a good quality random number generator,

```
sage: from Crypto.Cipher import AES
sage:
sage: def bitreverse(i):
....:              return eval('0b' + bin(i)[2:].rjust(128,'0')[::-1])
....:
sage:
sage: F.<x> = QQ[]
sage: K.<a> = GF( 2**128, name='a', modulus=x^128 + x^7 + x^2 + x + 1 )
sage:
sage: key    = "6261636f6e20696e206d7920736f6170".decode("hex")
sage: cipher = AES.new( key, AES.MODE_ECB )
sage: h      = eval( '0x' + cipher.encrypt( '\0' * 16 ).encode("hex"))
sage: H      = K.fetch_int( bitreverse( h ) )
sage:
sage: print( H.multiplicative_order() )
68056473384187692692674921486353642291
sage:
```

Figure 3.5: Calculating the Strength of a GCM AES Key

the probability of generating one of the weak keys is about 2^{-114}.

The software used to generate AES keys for use with GCM could be enhanced to check for, and reject weak keys by computing the order of the $H = EncryptAES(K, 0^{128})$ resulting from the generated key as described in [55]. If the order of H is less than 4099 then it is weak in the context of DLMS/COSEM over IP. The multiplicative order of an AES key used with GCM can be computed using sage[8] as shown in fig. 3.5.

3.5.2 Sophie Germain Counter Mode

Saarinen proposes a new stream cipher mode for AES called Sophie Germain Counter Mode (SGCM)[56] that replaces the field $GF(2^{128})$ that GCM uses, with the integers modulo $p = 2^{128} + 12451 = 340282366920938463463374607431768223907$, which is a Sophie Germain prime[1]. There are two weak subgroups with one element each and the remaining two are of order about 2^{127}[56]. The two weak elements are 1 and $p - 1$. The value 1 can be excluded and $p - 1$ never comes up because it is larger than 128 bits.

[1]Sophie Germain primes is a prime number p, such that $p = 2 \cdot q + 1$, where q is also prime

Chapter 4

DLMS / COSEM Protocol Overview

Here, we provide an overview and critique of the DLMS COSEM protocol to provide a foundation for the vulnerabilities described in chapter 7

4.1 Transport Level Protocols

DLMS / COSEM supports both HDLC[34] and IP[35] transport layers. The smart meter software provided supported HDLC, Transmission Control Protocol (TCP), and User Datagram Protocol (UDP) transports. It was possible to locate the HDLC standard but the IP standard was not available on the Internet. Fortunately, the IP wrapper protocol is described elsewhere and both are included in free software implementations[5][3].

4.1.1 TCP and UDP over IP

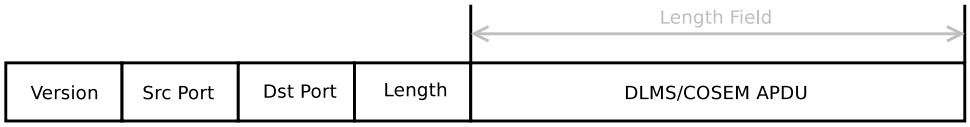

Figure 4.1: TCP / UDP Wrapper Format

Since TCP is a streaming protocol, there is no way to be sure where one Application Protocol Data Unit (APDU) ends and another begins, so a header indicating its length precedes each message. This wrapper protocol that applies to both TCP and UDP sessions ensures that an entire APDU is received before processing starts.

A wrapped message (fig. 4.1) starts starts with a a fixed 8-byte header consisting of four 16-bit (2 byte) integers in network byte order: version number, source port, destination port and payload length, followed by a DLMS/COSEM APDU of the length held in the payload length field header field. The source and destination ports are used in different ways by different implementations, but for the meters that we have tested, they are both always set to the value 1. The length field represents the length of the APDU being sent.

4.1.2 HDLC

The P0 ports (table 1.1) on some smart meters support IEC-62056-21[33] mode E, or DLMS / COSEM over HDLC. The protocol for requesting the meter to enter mode E is the same as is described in the standard but in practice, timing was an big issue in setting up the physical connection. The HDLC protocol specified in [34] is very similar to other descriptions of the protocol found via[61].

4.2 Session Establishment

Session establishment and tear-down uses the OSI Association Control Service Element (ACSE) protocol defined in ITU Telecommunication Standardization Sector (ITU-T) Recommendation X.227[58] and the document is freely available for download from the International Telecommunications Union (ITU) web site. The message sequence chart in figure 4.2 shows the ACSE messages in blue and the DLMS / COSEM APDUs in black. The dotted lines indicate optional parts of the protocol.

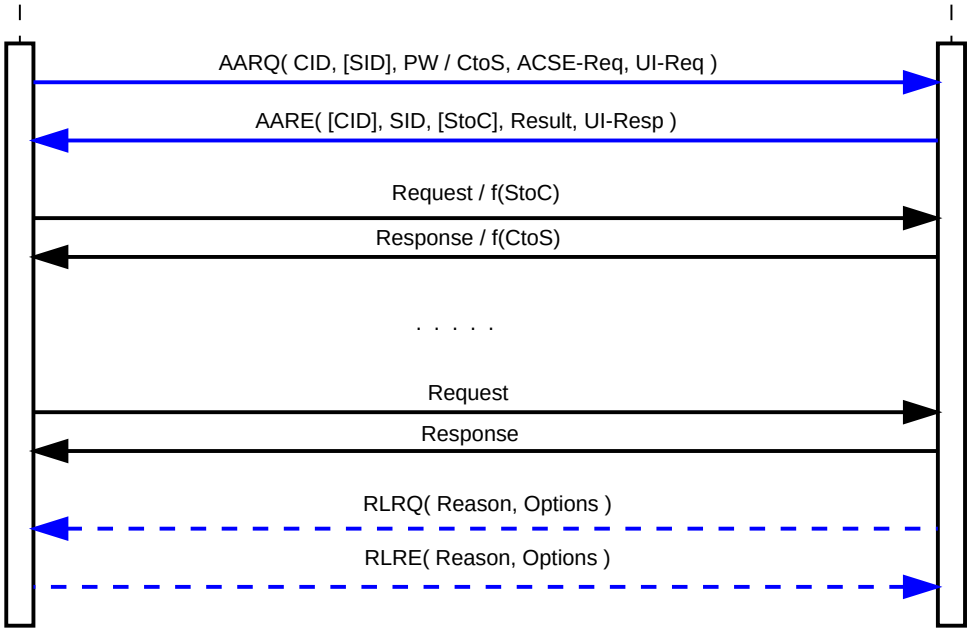

Figure 4.2: COSEM Message Sequence Chart

The ACSE protocol establishes and tears down an Application Association (AA) between a client Application Process (AP) and a server AP for a given *Application Context Name* (see table 4.1.) In DLMS / COSEM, the context name identifies whether an encrypted or plaintext application context is being requested (experimentally determined) and whether short name vs logical names will be used to address information on the server[9].

An AP is identified by an 8-byte AP Title where the first three bytes are used to identify an equipment vendor, followed by 0xff and then 32 bits that are required to be unique to a particular device[31]. The meters provided by our industrial partner used the 32 bits to encode the serial number of each device as a big-endian 32-bit integer and follow

OID	**Description**
2.16.756.5.8.1.1	Logical Name Referencing, Without Ciphering
2.16.756.5.8.1.2	Short Name Referencing, Without Ciphering
2.16.756.5.8.1.3	Logical Name Referencing, With Ciphering
2.16.756.5.8.1.4	Short Name Referencing, Without Ciphering

Table 4.1: Application Context Names

OID	Description
2.16.756.5.8.2.0	Lowest Level Security
2.16.756.5.8.2.1	LLS
2.16.756.5.8.2.2	HLS - Vendor Proprietary
2.16.756.5.8.2.3	HLS - MD5
2.16.756.5.8.2.4	HLS - SHA1
2.16.756.5.8.2.5	HLS - GMAC

Table 4.2: COSEM Authentication Mechanisms

the convention where a string like `0x414243FF0000002A` represents vendor code "ABC", serial number 42. The communicating devices exchange these values during their AARE negotiation (section 4.2.3.)

4.2.1 Low Level Security

A session being established can be authenticated by multiple methods indicated by an *Authentication-mechanism Name* Object Identifier (OID) (4.2) in an Association Request (AARQ) message. In addition to an authentication mechanism, a bit string called *ACSE User Requirement* is sent by the client AP in order to establish an authenticated session. The ACSE User Requirements field is as an Abstract Syntax Notation #1 (ASN1) bit string with the first bit indicating a request for *Authentication*[37][58]. If the first bit is high, then the *Authentication Value* field is verified in a manner consistent with the authentication mechanism indicated in the *Authentication-mechanism Name* field and if the authentication value is correct, the session is accepted, otherwise it is rejected. This mechanism is called Low Level Security (LLS).

There is another mechanism called "Lowest Level Security" in table 4.2 that is misleading because the mechanism is simply the lack of authentication and is indicated by the lack of an Authentication-mechanism Name and Authentication Value. The privileges associated with an unauthenticated connection are intended to be very low.

4.2.2 High Level Security

The message structure of ACSE is too rigid to support an authentication protocol that relies on a challenge/response mechanism so in order to support mechanisms like High Level Security (HLS) (and similar future mechanisms,) DLMS / COSEM violates its semantics and uses the Authentication Value field to carry a client to server (CtoS) nonce, and instead of replying back with a decision about authentication, the server always replies back with an accepted message and its own Authentication Value field that contains the server to client (StoC) nonce. Once the initial exchange is completed, the server moves the connection into a partially authenticated state where *Action Request* and *Action Response* messages are permitted to be exchanged with values computed depending on the authentication method chosen. Upon satisfactory completion of the *Action Request* that proves client identity to the server and *Action Response* that proves server identity to the client, the session is then elevated to a fully authenticated state. As is the case with any other message in DLMS / COSEM, the action messages associated with HLS can be encrypted or plaintext and the DSMR requires encrypted and authenticated communication.

Message Digest 5 (MD5)[53] and Secure Hash Algorithm 1 (SHA1)[17] HLS responses are nothing more than $f(nonce||authentication\_secret)$, where nonce is the challenge presented by the partner AP. The authentication secret in this context is a password and should not be confused with the authentication key in section 5.1.

HLS GMAC or (HLS level 5) responses are computed by applying the GMAC algorithm (see section 3.4) to an AAD that is the concatenation of the control byte 0x10, the 128-bit authentication key, and the nonce presented by the partner AP.

4.2.3 Application Parameter Negotiation

In addition to the generic pieces of ACSE that apply to any application using it, there is a *User-Information* exchange included to negotiate specific details of the application level protocol that follows the authentication. The client sends an *xDLMS-Initiate.request* APDU and the server responds with an *xDLMS-Initiate.response* APDU. The APDUs can be either plaintext or ciphered, though it is unclear how the distinction is made, apart from perhaps checking whether the message is either a well-formed ciphered APDU or a well-formed plaintext *xDLMS-Initiate* request or response APDU. When manipulating the encrypted user data setting, the only difference in the resulting AARQ message is the content of the User-Information field. When comparing the plaintext of an encrypted User-Information field with the content of a plaintext User-Information field from traces produced in the lab with a real meter and its official client software, the two were identical.

The xDLMS-Initiate request and response APDUs contain a number of parameters related to the operation of the application level protocol. The client can provide a dedicated key for APDU encryption, a parameter to suppress transmission of an Association Response (AARE) message from the server, protocol version, a 16-bit integer declaring the the

maximum size of APDU supported by the client, and a bit field indicating functionality being requested. There is an optional proposed quality of service field that is listed as not being used in [22] but is useful for the attack in section 7.4.2.

The server then responds (if response-allowed is enabled,) with a similar message that contains a negotiated protocol version number, bit field indicating negotiated functionality, a 16-bit value indicating the maximum APDU size acceptable to the server, and vaa-name, which is not explained in detail but is 0x0007 when logical naming is specified by context name and 0xFA00 when short name referencing is specified. Although negotiated quality of service is not supported in the current protocol version, when presented with a proposed quality of service in an xDLMS-Initiate.request message, the response from the test meter also contained a negotiated quality of service field. Nothing appears to change, apart from the size of the response APDU, which increases its size.

4.2.4 Association Tear-Down

There are three messages that can be used to tear down an AA: Release Request (RLRQ), Release Response (RLRE), and Abort (ABRT), which we discuss here for completeness. The RLRQ and RLRE APDUs are used to perform a normal orderly release of an AA, though an AA may also released by closing the connection at a lower layer such as HDLC or TCP, as is done by the official client software used in the lab. The ABRT message is used for abnormal termination triggered by an unrecoverable error.

Disconnect reasons *normal(0), urgent(1), and user-defined(30)* are supported for RLRQ and *normal(0), not-finished(1), and user-defined(30)* for RLRE. For reasons not clearly described in [22], an optional User-Information field for an xDLMS-Initiate.request and xDLMS-Initiate.response APDU is included in both. Neither [22] nor [37] explain how the messages are interpreted in that context.

4.3 COSEM Commands and Objects

In this section, we discuss commands and objects in the context of DLMS / COSEM, but we take some liberties with the nomenclature in the interest of clarity.

4.3.1 Commands

The characterization of commands like "Get," "Set," "Action," etc. as Services can be easily confused with "Service" as used in general-purpose computing, so we call the command itself a *request* and the response to the command a *response*. Furthermore, we provide only a short description of commands because their format is documented and available in [37] ,[36], and the ASN1 specification included in [5].

4.3.2 OBIS Codes

Commands act upon objects and objects are references to data or more abstract structures like an AA. The objects are are identified by 6-byte Object Identification System (OBIS) codes that are in turn organized into ranges that represent various standardized areas like communication, gas metering, electricity metering, etc. In addition to the main functional sections with objects likely to be universally applicable, sections of the address space are reserved for national level standards and proprietary extensions that any manufacturer can make use of. Details of the organization and definition of the standardized codes, such as those used for reading power utilization are defined in IEC-62056-61[38] and a machine readable list is available from the DLMS User Association web site[1] as a Microsoft Excel spreadsheet[1].

4.4 COSEM APDU Format

The APDU format is defined as an ASN1[39] syntax specification of approximately 18 pages in [37] and is also published in the source code for jDLMS [5] (jDLMS). ASN.1 Complete[46] provides a good introduction to ASN1 for those not already familiar with it.

4.4.1 ASN1 and A-XDR

On the surface, it seems straight-forward to implement such a protocol, except that DLMS / COSEM uses the Adapted eXtended Data Representation [32] (A-XDR) encoding scheme in addition to Basic Encoding Rules [39] (BER), which is used in ACSE. A-XDR is a relatively obscure ASN1 encoding scheme, specific to electrical power industry that is said to have a more compact representation than competing encodings. It achieves its compactness in part by removing features like BER's self-describing encodings that allow a message to be decoded into basic data structures without the need of an explicit syntax specification.

Since A-XDR does not provide such self-describing data structures naively, the part of the APDU syntax that describes data was written in such a way as to explicitly include type information.

Possibly the greatest advantage of messages that based on an ASN1 syntax specification is that the syntax specification, which often includes constraint information, can be directly compiled into e.g. C source code that can produce and consume well-formed APDUs without a developer needing to directly implement them. The automation reduces the likelihood of mistakes in implementations, speeds development, and frees developers to concentrate on issues directly related to the purpose of their application.

The Fraunhofer Institute's jASN1[6] compiler generates Java classes that process A-XDR

[1] in xlsx format and readable outside excel

encodings of an ASN1 message syntax specification, and appears to be the only one of its kind. The jDLMS[5] plaintext DLMS / COSEM library uses the compiler to produce all the code it uses to process DLMS / COSEM APDU and thus, any time a change is made to the specification, a developer need only replace the ASN1 syntax specification with the modified one, re-compile, and modify their business logic as necessary to support the new or modified functionality.

From a security perspective, the choice to add a new encoding scheme increases program complexity and attack surface in exchange for a more compact message encoding. While highly mature implementations like jDLMS are clearly possible, it is not every development team that has the resources to build their own ASN1 compiler to support a new encoding scheme that they will use for one protocol. The more likely decision taken by a team with limited resources would be to write ad-hoc code to process the specific APDUs required for their implementation. That appears to have been the approach taken by the GuruX project[3].

The International Electrotechnical Commission (IEC) report of patents covering their standards was consulted and surprisingly, there were none listed that covered the A-XDR scheme, which lends support to the conclusion that the DLMS User Association required the encoding scheme because they believed the efficiency gains justified the added complexity and attack surface.

4.5 Authorization

Authorization in COSEM can be based on how authentication was performed, partner AP title, operation, and object on which the operation is to be performed. Security policy details are left to developers, customers, regulators, etc. to decide on.

Chapter 5

COSEM Use of Cryptography

The purpose of this chapter is to document the information acquired from a range of public sources and verified by laboratory experimentation. We refer to these details in chapter 7 and hope that they will stimulate open discussion of security implications of using the protocol.

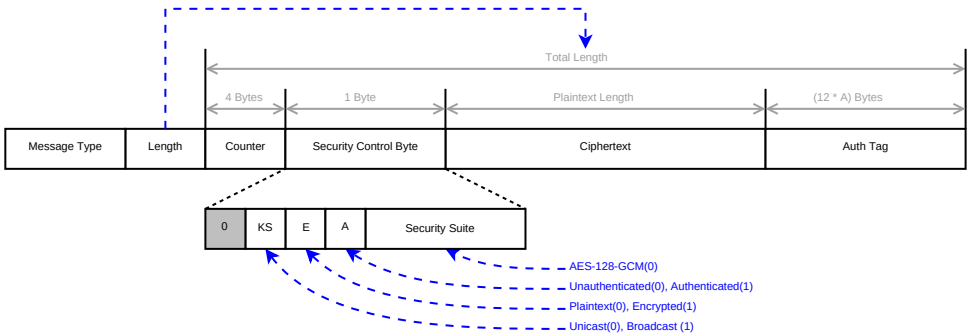

Figure 5.1: Diagram of Ciphered COSEM APDU

In this chapter, we discuss the ciphering and message authentication methods required by the DSMR[27]. The information presented in this chapter assumes that what information is contained in [22] is accurate and is otherwise based on traffic observed between certified client and server implementations and other unprivileged information.

The method for APDU ciphering was deduced from several sources [21] [60] [31] [16] and experiments supported by having access to a pair of smart meters whose keys were known to us. The details were also combined with old versions of the plaintext DLMS / COSEM protocol specification to fill in missing information.

The DLMS User Association recently published a slide deck[43] from their security training course at the 2014 European Utility Week conference in Amsterdam, that includes a very detailed illustration that validated what we had already learned about the cryptography from other public sources. The slides also mention that a new protocol version may include two new cipher suites and data compression support.

5.1 Key Material and Security Claims

Since our purpose is to evaluate the security of the DLMS / COSEM protocol, let us start by defining key material and making some implicit security claims regarding the protocol explicit. The following is a list of key material that can be assumed not to be known to an attacker unless otherwise stated:

Master Key	An AES key programmed into a meter's firmware at the time of manufacture that is used to change ciphering and authentication keys used for communication.
Ciphering Key	The 128-bit AES key used for GCM encryption of application level APDUs.
Authentication Key	The 128-bit authentication "key" is 16 byte (128-bit) bit string that is included as AAD when a message in need of an authentication tag is being encrypted using GCM. Since it is called an authentication key and since it is used in the calculation of authentication tags in DLMS/COSEM then it seems fair to conclude that the purpose of adding the key to the AAD was as proof that a message was produced by a protocol participant with knowledge of the string. Please note that the security of message authentication in GCM does not depend on the secrecy or presence of AAD.

Let us now propose the following, hopefully uncontroversial security claims for the DLMS / COSEM protocol:

(a) messages cannot be decrypted without knowledge of the correct 128-bit ciphering key.

(b) Both the authentication and ciphering keys are necessary to produce a valid DLMS / COSEM authentication tag.

5.2 APDU Ciphering

Ciphered DLMS / COSEM APDUs have a field that indicates the cipher suite in operation. The specification being discussed here specifies a suite zero, which makes use of a pre-shard 128-bit AES key, a pre-shard 128-bit authentication "key", and makes use of the GCM[47] authenticated block cipher streaming mode and is required to be used by DSMR[27]. Per [43], two new suites are likely to be added to a future edition of the protocol that may address some of the shortcomings of suite zero, but without any documentation, no meaningful discussion of their merits is possible. Please see chapter 7 for a full explanation of protocol vulnerabilities in the current version.

Message	Plaintext	Global	Dedicated
Get Request	192	200	208
Set Request	193	201	209
Event Notification Request	194	202	210
Action Request	195	203	211
Get Response	196	204	212
Set Response	197	205	213
Action Response	198	206	214

Table 5.1: COSEM Message Identifiers

5.3 Ciphered APDU Message Headers

The DLMS / COSEM message syntax specification defines one main APDU type called *COSEMpdu*, which is a choice of several different types of APDUs, most of which have tags associated with them. This applies to both ciphered and plaintext APDUs. Each choice is identified by a tag and the ciphered APDUs are of the type *IMPLICIT OCTET-STRING* and start with an 8-bit identifier followed by the octet string's length – the encoding of the length of the octet string follows the A-XDR encoding scheme and appears to follow the same short form encoding as BER.

If the length of the octet string exceeds 127 bytes then the most significant bit of the first octet in the length will be high and the remaining bits indicate the number of bytes used to indicate the length of the field. Those bytes are interpreted as an n-byte big-endian unsigned integer[1].

Ciphered APDUs (see fig. 5.1) are identified by the tags that indicate message type and the type of key used to encrypt the payload APDU. The key types of are global ciphering, which uses a long-lived pre-shared key and dedicated ciphering that uses a one-time session key that is established in the ACSE exchange (section 4.2.) The dedicated mode is not allowed by DSMR and McGrew and Viega strongly caution against the use of long-lived keys in GCM in [59]. See table 5.1 for the complete list of plaintext, globally ciphered, and dedicated ciphered message identifiers.

5.4 Security Control Byte

The first byte of the actual octet string holding the ciphered APDU is the security control byte, whose fields are broken out in the following table 5.2.

The meaning of the fields are as follows:

- The reserved field is of course reserved and set to zero.

[1] this encoding may lead to exploitable code but control flow attacks are out of scope

Bit 0	Bit 1	Bit 2	Bit 3	Bits 4-7
Reserved (0)	Key Set	Encrypted	Authenticated	Security Suite Identifier

Table 5.2: Security Control Bits

- *unicast (0)* and *broadcast (1)* are listed in the message syntax specification but the value of this parameter appears to always be zero since no values other than seen values 0x10 (authenticate only), 0x20 (encrypt only), and 0x30 have been observed and [31] states when discussing the security control parameter (byte), that "there are 3 variants: *Authentication only*, *Encryption only*, and *Authentication and Encryption*." While what is stated there is not authoritative, it is the statement of a company that sells a certified DLMS / COSEM library, so the insight does carry some weight.
- The Encryption and Authentication bits indicate whether encryption, authentication, both, or neither is enabled. The neither case, oddly enough, is allowed and just results in twice the header overhead and increased likelihood of implementation errors that allow replacement of actually encrypted messages with messages that are just encoded in a slightly different way.
- The security suite identifier is the context in which the encrypted APDU should be interpreted. Suite zero is the most common and to my knowledge, only suite in operation in the Netherlands. The statement about security control parameter in [31] supports that belief.

5.5 Frame Counter

The next four bytes in a ciphered APDU are a big-endian, unsigned 32-bit integer that should represent the total number of frames transmitted by the AP sending the message since the last time its global ciphering key was updated.

5.5.1 Computing Ciphertext Length

The length of the octet string is used to determine ciphertext length. The message can be no less than the header length $L_h = 5$ bytes long (1 byte for the security control byte and four bytes for the frame counter. The length field, which we shall call L_m, accounts for the entire enciphered message. Let the length of the ciphertext and authentication tags be L_c and L_a, respectively.

If the encryption bit is low, then the ciphertext section is replaced with the plaintext, but the length of both are L_c. If the authentication bit is high, then a 12 byte ICV is included at the end of the message and $L_a = 12$, otherwise: $L_a = 0$. The length of the entire message, $L_m = L_h + L_c + L_a = L_c + L_a + 5$ and the ciphertext length $L_c = L_m - L_a - 5$.

5.6 DLMS / COSEM use of GCM

The GCM ciphering process prefers an IV of 96 bits, but allows an IV of arbitrary size. COSEM opts for the ideal preferred IV length and defines it as the concatenation of a 64-bit AP title followed by a 32-bit, unique frame counter. The titles of participating APs are exchanged in the ACSE exchange that starts an AA described in section 4.2.

5.6.1 Authentication Tags

The AAD is the concatenation of the security control byte, followed by the 128-bit authentication key. The use of the 128-bit authentication "key" is not called for by GCM and adds very little security, if any, over standard GCM.

The encryption operation results in ciphertext and a 128-bit ICV as shown in fig. 3.2. COSEM truncates the 128-bit MAC from GCM to 96 bits. The truncation may well reduce the security of the hash slightly below 96 bits of uncertainty for large messages (section 3.5.)

5.6.2 Encrypted Messages

Based on whether encryption, encryption and authentication, both, or both are performed, the fields shown in fig. 5.1 are filled as indicated in section 5.5.1.

Chapter 6

Laboratory & Tools

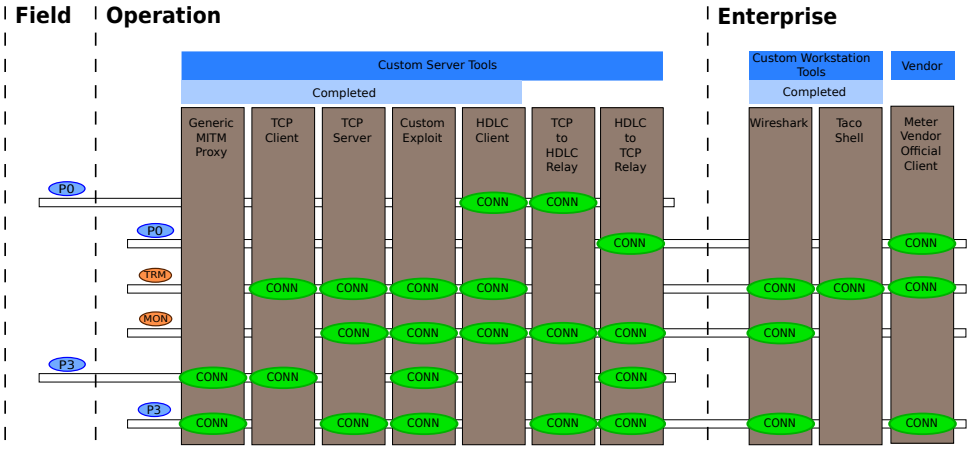

Figure 6.1: Overview of Laboratory Components

6.1 Conceptual Organization

Components of our experimental setup are organized across zones, and communication ports, much like the DSMR model of the Dutch AMI and the AMI Reference Model[14].

To avoid what would otherwise be a large number of connections between components, each type of communication is expressed as a bus line across all components instead of illustrating individual connections. Naturally, the SGAM domain is the customer premise because we are testing smart meters.

We add two new logical channels Remote Monitoring (MON) and Interactive Terminal (TRM) to the standard DSMR communication ports in table 1.1 because in a laboratory, we must be able to express interaction via an interactive terminal (TRM) and transmission of communication communication traces for real-time dissection at an evaluator desktop. Each application / device represented in the diagram is represented as a rectangle laid over several bus lines representing a particular channel.

Enterprise	Evaluators look at systems from a high level perspective, develop new ideas about exploitation, and make use of a shared laboratory setup via applications in the server zone to test them out. The primary tools in this zone are the Wireshark protocol analyzer with custom protocol dissectors and the *taco shell* that provides a command-line interface to cryptographic and binary level protocol elements.
Operation	Our operation zone contains components that provide services for accessing field or station level resources, though no station level is present in our actual setup. The separation of the operational zone facilitates efficient sharing of limited resources. Applications in the server zone are naturally the physical infrastructure for performing testing and the various Man in The Middle (MITM) proxies, protocol-level bridges (not yet developed), and a repository of previously developed exploits.
Field	The field zone simply contains the devices under test.

Table 6.1: SGAM Zones in Laboratory Context

See table 6.1 for a concrete description of the SGAM zones in the context of our testing methodology.

6.1.1 Physical Laboratory Configuration

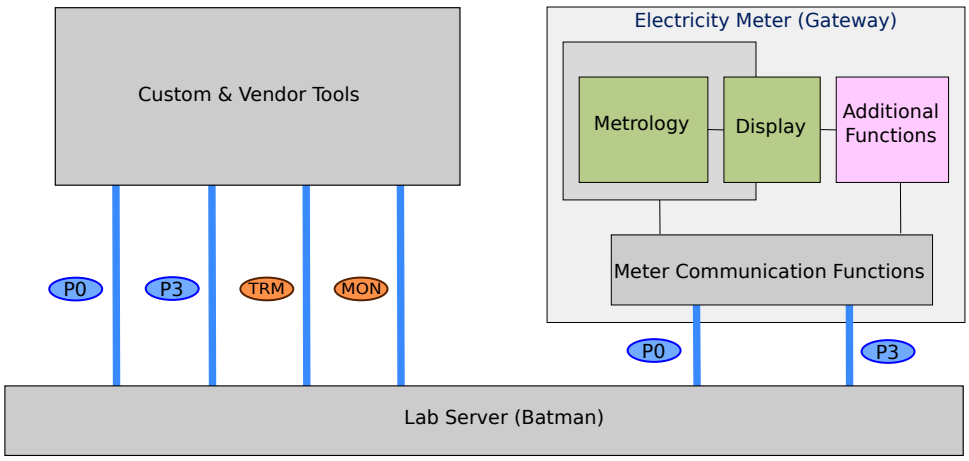

Figure 6.2: Laboratory Communication Topology (Adapted from [14])

There is a physical Ubuntu server in a lab that provides the primary connectivity between client / vendor tools and a physical smart meter. The server is connected physically to the IEC 62056-21 optical port (P0) of the meter and connects to the TCP/IP over GPRS (P3) meter interface via the open Internet.

The vendor-supplied management software for the smart meters being evaluated is in-

Item	Source
Smart Meters	Industrial Partner
DLMS / COSEM Authentication Keys	Industrial Partner
DLMS / COSEM AES Global Ciphering Keys	Industrial Partner
Vendor Default LLS Access Password	Documentation
Laptop Computer	Purchased
Client Software with Working Configuration	Purchased
USB IEC 62056-21 Optical Transceiver	Purchased
GPRS SIM with Static IP Address	Purchased

Table 6.2: Day One Experimental Setup

stalled on its own laptop running Windows XP, primarily because there is only one license and it needed to be shared among multiple users. Workstations running Debian were used for software development and interactive evaluation.

6.2 Materials Provided

The experimental setup for this research consisted of two smart meters with associated ciphering and authentication keys, a Universal Serial Bus (USB) IEC 62056-21[33] optical transceiver, a copy of the official management software from the meter vendor that was already configured to work with the meters using the provided key material and the default access password for the meters. The vendor software does not directly state whether LLS or HLS are being used, so due to incomplete information, it was initially assumed that HLS but upon closer examination, it was clear that in fact LLS was being used. See table 6.2 for the sources of these items.

The rules of engagement for the project were set to restrict the research to information that could be obtained without privileged access. The so-called "Green Book"[21] that documents version 7 of the DLMS/COSEM protocol was not provided and since we were unable to obtain a copy of the document under the rules of engagement, the research had to proceed without it.

Reference implementations of both client and server protocols were available in the form of the purchased administration software and an actual meter provided by an industry partner, we could easily validate hypotheses about the protocol and in particular, the details of the cryptography. Though access to an official meter used by a grid operator provides a high level of credibility to our practical results, evaluation boards with similar protocol support were available from Microchip and ST Micro-electronics were available.

6.3 Meters

Two smart meters based on the DLMS / COSEM Green Book 7^{th} Edition protocol were provided by an industrial partner. The authentication and encryption keys for the two meters were also provided to aid in the evaluation. The particular meters still had their default authentication passwords set, so those were not required.

6.4 Optical Transceiver Cable

The most immediate need to be met in the project was to be able to communicate with the meter in a meaningful way and observe communication. This was done with a commercially available optical interface cable, the details of which are specified in section 5 of [33]. The communication is V.28 serial with infrared used as the physical interface. The presence of a sufficiently intense infrared light source between 800nm and 1000nm in wavelength indicates a logic zero and the absence of the light source indicates a logic one. The particular purchased cable presented itself as a Future Technology Devices Internations, Ltd .[2] (FTDI) FT232 serial Universal Asynchronous Receiver / Transmitter (UART) and the client software on the Windows laptop used in most experiments seemed to make no distinction between the special interface cable and any other serial port.

An early attempt was made to create a proxy between the optical interface and a standard RS-232 serial port, resulting in some readable information, followed by data that did not match what the client claimed to be sending. The problem was due to a negotiation step where the meter instructs the client software to increase its baud rate to 9600 baud. The proxy could have been corrected with the addition of code that recognized the instruction but long before that information had surfaced, it was discovered that the client software could dump raw communication traces that we could transform into pcap formatted files for use in Wireshark (section 6.7.)

The cable was used in the first couple months of the research but given that we had chosen to focus on the application level protocol rather than the low-level protocol details that were likely only to result in product- or library-specific attacks, we switched to the more easily implemented DLMS / COSEM over TCP/IP as soon as was possible, suspending development of the serial proxy in favor of a more generally applicable TCP proxy.

6.5 Vendor-Supplied Client Software

The company sponsoring this work purchased a copy of the official management software for the meters in question from the device's vendor and arrived at a working configuration before the start of the project.

6.6 cosemCrypto Library

The library implements the basic cryptographic functions used in DLMS / COSEM as well as more specific functions used to facilitate development of exploit code in Python.

6.7 pcapify_hdlc.py

The *pcapify_hdlc.py* script takes a text file filled with lines of American Standard Code for Information Interchange (ASCII) encoded hex digits and packages them into a pcap formatted file for use in Wireshark. The tool was created so that we could write Wireshark protocol dissectors to process the communication traces produced by the proprietary client software that was purchased for this project.

6.8 Sage Math

Sage[8] is an interactive mathematics program that has convenient support for computation over finite fields and has been helpful for computations like the example in fig. 3.5.

6.9 Taco Shell

The *Taco Shell* is an interactive shell, named after delicious food and written in Python that exposes the functions in *cosemCrypto* for interactive use, most input and output is in ASCII encoded hexadecimal, matching the other interactive tools.

As a convenience, the Taco shell has variables for encryption keys so that the keys need not be typed in every time a command is entered. There are set and get commands to set and display variables like cryptographic keys.

The primary purpose of this tool is to act as something of a "Swiss Army Knife" for manipulating DLMS / COSEM messages from the various other custom tools, including the interactive client / server utilities and of course Wireshark, which has the ability to place a copy of packet or dissection data into the system clipboard, which can then be "pasted" into taco for decryption or other processing.

If, for example, Wireshark has an encrypted packet then it can be extracted, pasted into Taco for decryption.

The commands for taco listed in the following subsections.

6.9.1 decrypt

Decrypts and authenticates DLMS cryptograms that are both encrypted and authenticated. If the authentication tag is not available, the decryption will be reported as invalid, because there is trivial to replace any given known plaintext with anything you

like. Arguments are as follows:

All parameters are expected to be hexadecimal strings with no white space.

auth_tag is the authentication tag generated for a message. It is a mandatory parameter to underscore its importance. When deploying an IEC/COSEM/DLMS meter, it is never the right choice to communicate with only encryption enabled.

```
taco: decrypt  abaad8df7f10562d 313233ff000000010000002b  afcb50225e64847c5e4a876f

 Ciphertext.............. "abaad8df7f10562d"
 Initialization Vector.... "313233ff000000010000002b"
 Auth Tag................ "afcb50225e64847c5e4a876f" (Passed)
 Plaintext............... "cafebabedeadbeef"
```

6.9.2 encrypt

Encrypt a plaintext string using the AES-128-GCM stream cipher, producing a ciphertext and matching authentication tag, compatible with COSEM/DLMS encryption. The function assumes that the message is both authenticated and encrypted because that is the only case in which an authentication tag would be relevant.

```
taco: encrypt cafebabedeadbeef  313233FF000000010000002B

 Plaintext............... "cafebabedeadbeef"
 Initialization Vector.... "313233ff000000010000002b"

 Ciphertext.............. "abaad8df7f10562d"
 Auth Tag................ "afcb50225e64847c5e4a876f"
```

6.9.3 exec

Utility function that takes a Python command as a parameter and executes it, returning the result. It is a convenience function that is often used for conversion from hex to decimal and back.

6.9.4 fcs

Takes a binary string and computes an HDLC frame-check sequence, also known as a CRC-32 checksum.

```
taco: fcs cafebabedeadbeef
96d7
```

6.9.5 hdlcWrap

This function takes receive and transmit sequence Numbers and a payload, outputting a well-formed HDLC frame with correctly computed checksums. The output is intended to be used with the HDLC client program. The output is as follows:

```
taco: hdlcWrap 0 0 c90d200000001b9710321f410a4d89

 Original.......... c90d200000001b9710321f410a4d89
 Wrapped........... 7ea01c02230310fd5be6e600c90d20...89135c7e
```

6.9.6 hls34

Computes HLS (MD5 and SHA1) response function

```
taco: hls34 md5 deadbeef taco

Hash:                   md5
Challenge:              deadbeef
Password:               taco
Response:               1fc1d534732c1302fd2e2e79d680d3b5
```

6.9.7 hls5

This function computes an HLS-GMAC response

```
taco: hls5 deadbeef 42 313233ff00000000

Hash:                   gmac
Challenge:              deadbeef
Title:                  313233ff00000000
Counter:                42
Response:               10421ff3b150e0661ec831da4d2e
```

6.9.8 pducrypt

This function performs encryption and encapsulation in one step, producing well formed encrypted as well as encrypted / authenticated APDUs. The output also includes pre-computed TCP / UDP headers.

```
taco: pducrypt 313233 27 201 deadbeef

 Plaintext.......... deadbeef
 Raw Ciphertext..... 758ef7ae
 Auth Tag........... 0f510d342af21fa5ee14855b
 Encrypted.......... c909200000001b758ef7ae
                     000100010001000bc909200000001b758ef7ae
 Both............... c915300000001b758ef7ae0f510d342af21fa5ee14855b
                     0001000100010017c...510d342af21fa5ee14855b
```

6.9.9 tcpUnwrap

This function extracts a message from it's TCP header and displays information about what it just decoded:

```
taco: tcpUnwrap 0001000100010017c9153000000...1fa5ee14855b

 Wrapped............ 0001000100010017c91530...42af21fa5ee14855b
 Payload............ c915300000001b758ef7ae0f510d342af21fa5ee14855b
 Version............ 1
 Source............. 1
 Destination........ 1
 Length............. 23
```

6.9.10 tcpWrap

This function takes a hex string and wraps it inside a TCP header as follows:

```
taco: tcpWrap 1 1 c915300000001b758ef7ae0f510d342af21fa5ee14855b

 Original........... c915300000001b758ef7ae0f510d342af21fa5ee14855b
 Wrapped............ 0001000100010017c915...510d342af21fa5ee14855b
```

6.10 Interactive Client / Server Tools

the two tools allow a user to play the role of either a DLMS/COSEM client or server by interacting with either the client or server program from a terminal. Whatever the user sends to the terminal in ASCII encoded hexadecimal is converted to binary and sent to the network end of the connection and vice versa.

The tools can be used manually, to interact with e.g. a meter to test ideas or to replay messages or the two can be used in concert to allow the evaluator to manually perform MITM attacks.

6.10.1 MITM Proxy

This framework is used to implement a MITM based proofs of concept. Several proxies have been built out of the basic proxy for tasks ranging from simply observing traffic, the compound command injection proxy described in section 7.4.3.

6.10.2 HDLC Client

The HDLC client contains enough code to negotiate with a meter to establish a mode E connection followed by a message to the meter that makes it accept DLMS / COSEM APDUs embedded inside of HDLC info packets. Just like the TCP client and server, interaction is via command line with binary data given to the client in hexadecimal.

Once connected, it sends exactly the binary messages given on standard input to the meters P0 port, reading the responses until a one second timeout has expired, and printing them to standard output as hexadecimal.

In addition to the messages going to standard output, every message sent or received is sent over a UDP connection to a configurable IP address and port 4060. Wireshark is configured to pass UDP packets on port 4060 to the HDLC dissector (section 6.11) for real-time dissection.

Generally speaking, the HDLC client executes on a server in the operational zone that is connected to P0 of the lab meter at all times. Anyone who needs access to the meter can use it and have the protocol traces automatically delivered to Wireshark in their office. This tool is the precursor to the TCP to HDLC proxy that will allow us to run all the TCP based proof of concept code and proxies against the P0 port without modifying them.

6.11 Wireshark Dissectors

The tool development process started with the development of a Lua[4] based Wireshark dissector[11] for COSEM over HDLC because although there are several HDLC

based protocols supported by Wireshark (ISDN, X.25, etc.,) those dissectors are largely monolithic. The Cisco HDLC dissector could be built on, but Cisco HDLC is different enough from the HDLC variant used by DLMS / COSEM that it was not usable without significant modification.

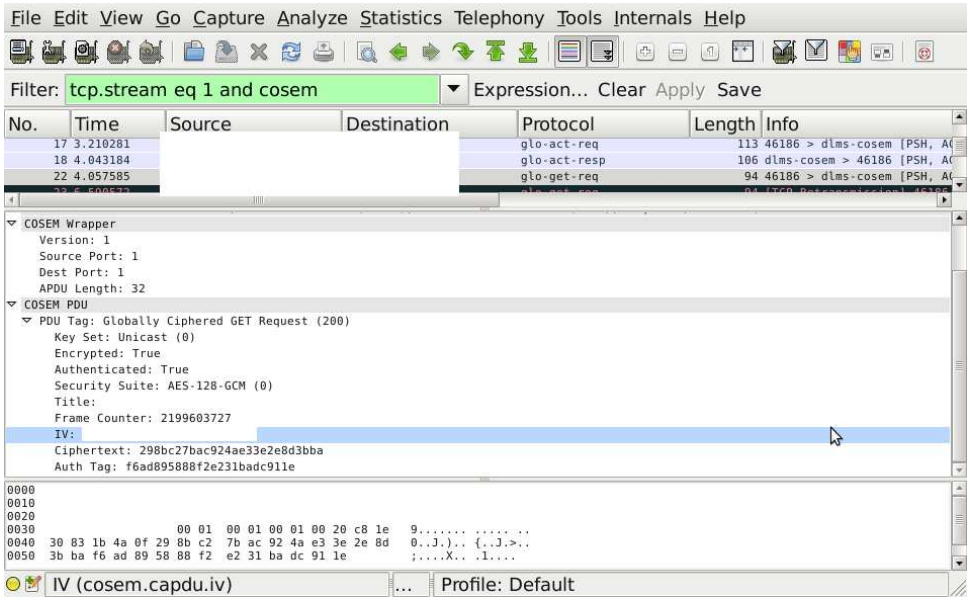

Figure 6.3: COSEM Dissector for Wireshark

6.11.1 Packet Capture

A protocol dissector is only useful when Wireshark has a facility for capturing communication live or loading the capture from a file. Since the serial proxy was not in operation and there was no automated method of feeding the HDLC over v.28 serial traces into Wireshark, an intermediate tool called `pcapify-hdlc.py` was written to convert traces provided by the proprietary client software into *pcap* files containing individual HDLC frames labeled as one of several frame types reserved for private use, namely "USER0."

6.11.2 Dissector Development

The Wireshark dissector development served as a repository for information collected about COSEM or HDLC. New format information was used to refine the dissectors, dissector changes were in turn validated against known messages.

Once the IP connection was up and running, the DLMS / COSEM (*cosem*) and HDLC (*hdlc*) dissectors were split apart and a TCP and UDP encapsulation protocol dissector (*wpdu*) was written and registered with port numbers 4059/tcp and 4059/udp. Since *cosem* was a separate module, it was a simple matter to layer it on top of the *hdlc* and *wpdu* dissectors. All further additions to the *cosem* protocol dissector then worked in HDLC, TCP, and UDP.

6.11.3 Features

The three dissectors are built to break messages up into fields based on the content of a single APDU or the associated session, in the case of TCP or UDP where Wireshark is already capable of tracking sessions. The features of the dissector are focused on security, so cryptographic information is well covered, where only the plaintext message types and generic data are as complete as was required to read e.g. HLS messages while verifying hypotheses about how the authentication method worked.

Fields extracted from APDU's in communication sessions are searchable in the same way that fields of other dissectors are searchable within a Wireshark session. Field searches were helpful when examining sessions with multiple requests and responses. For example: when validating hypotheses about how APDU ciphering and authentication worked, searching for AARQ and AARE messages was critical because they contain half of the information required to determine the IV needed to decrypt / authenticate a given message within a session. Tables A.3, A.1, and A.2 in appendix A contains a list of selected fields and their meanings from each protocol dissector.

6.11.4 Ciphered APDUs

The protocol dissector does not directly support encryption, decryption, or authenticity verification for ciphered APDUs but instead facilitates decryption by external programs by tracking the title of participants in the protocol, generating the correct IV for decryption based on frame counter, security parameters, and participant title. The dissector also separates ciphertext and authentication tags into different fields. Each can be copied from Wireshark and pasted into *taco* for decryption / manipulation.

Chapter 7

Vulnerabilities

The process of coming to understand the COSEM protocol, observing communication between client and server, and developing tools to enhance that understanding has led to some significant, exploitable vulnerabilities. When starting the project, a choice was made to make the results more realistic by limiting the information available to only those sources freely (or accidentally) available on the Internet and analysis of traffic between commercially available meters and software with known keys. We have identified what appear to be design flaws in the protocol itself as well as vulnerabilities found in the evaluation of two smart meters (a single-phase and a three-phase variant, both with shutoff relays installed.)

7.1 Protocol Vulnerabilities

The following protocol vulnerabilities do not necessarily result in compromise by themselves, but they serve to weaken the overall system so that it is less able to cope with easily introduced implementation flaws that might not otherwise have been exploitable.

7.1.1 Separable Authentication and Ciphering

Alone, stream ciphers based on plaintext XOR keystream model that GCM and other counter-based ciphers use are vulnerable to bit-flip attacks, particularly in settings where an attacker is likely to be able to predict the plaintext contained in a message.

The presence of an authentication tag is indicated in each ciphered DLMS / COSEM APDU in plain text, so since message authentication is optional, there is no way to securely ascertain whether a partner AP intended it to be authenticated of not. As a result, if a device is willing to accept messages without authentication then it must be assumed that an adversary who wishes to manipulate messages will remove their authentication tags as described in 7.3.2. If GCM were used correctly[47] in DLMS / COSEM and authentication tags were not allowed to be disabled, then the downgrade

attack would not be an option.

7.1.2 Information Leakage

Since an enciphered APDU contains an encrypted copy of the plaintext APDU being communicated, it would be sufficient to merely indicate that the APDU is encrypted and on which key (global or dedicated). Unfortunately, rather than adding two new message types: ciphered message utilizing global or dedicated keys, ciphered counterparts for each of the existing message types were added. The information leaks both the purpose of the message and the content of the first few bytes of the payload because every e.g. Get request must always start the same way.

Not only is the information leakage unnecessary, but the meters tested in the lab showed no evidence that they did anything with the information except determine how to decrypt the message and then pass the result on to be processed. Changing message type from "globally ciphered get request" to for example "globally ciphered get response," had no noticeable effect on behavior, for example.

7.1.3 HLS Server Impersonation

The method by which a server computes a response to a client challenge is identical to the method by which a client responds to a servers challenge[12]. Given the right circumstances, a malicious server may succeed in establishing a mutually authenticated AA by replaying the client's title, nonce, and hashed responses. Such an attack would require the ability to read and respond to the HLS responses coming from the legitimate client. The most likely situation that meets those requirements would be a plaintext session, which is not allowed by DSMR.

7.1.4 HLS Off-Line Dictionary Attack

When a plaintext (or otherwise decipherable) session is authenticated using HLS with MD5 or SHA1, an off-line dictionary attack is possible. To perform this attack, an adversary must acquire a valid HLS response and its corresponding nonce (section 4.2.2.)

Since the client sends its HLS response first, collection of a valid HLS response can come from observation of an accepting conversation between legitimate client and server or the attacker can implement the server side of the ACSE protocol that way.

Once the valid HLS response is acquired, the attacker has a nonce n and $h = f(n||password)$, where only the password is unknown and the attacker knows how to compute f on their own. The dictionary attack is performed by generating a dictionary of trial passwords t, comparing $f(n||t)$ with the h that has been acquired from a valid client and/or server.

Recalling that the nonce n used by the client to compute its $h = f(n||password)$ is

presented by the server, an attacker acquiring the valid nonce by means of the server impersonation attack would do well to always present the same nonce. If the nonce used for the off-line attacks can be held constant then Time / Memory Trade-Off (TMTO) attacks like rainbow tables used by OPHCRACK[50] will be helpful for attackers interested in collecting passwords on a large scale.

7.1.5 Responses Not Tied to Requests

Responses from DLMS / COSEM requests do not appear to provide any unpredictable information that ties them to the accompanying request. If a request sent to a server is replaced in an MITM attack and the data returned from both requests contains the same data type then there is no way for a client to detect that the response received back from the server is the one they originally requested.

7.1.6 MAC Forgery Without Authentication Key

DLMS / COSEM uses the GCM authenticating stream cipher mode for AES with a 128-bit key for APDU ciphering. The security of message authentication in GCM itself is based solely on the secrecy of it's AES encryption key without need of Additional Authenticated Data (AAD). In fact, the stated purpose of AAD is to include related plaintext data in e.g. the network packet carrying a GCM encrypted and authenticated message.

If an attacker in possession of ciphering key K can make partial use of the incremental MAC feature of GCM (section 3.4.1) to use a ciphered DLMS / COSEM APDU with a particular number of ciphertext blocks l to produce a valid authentication tag for a different message with the same number of ciphertext blocks ($128l$ bytes or less) without knowing the authentication key. The limitation is due to the truncation of DLMS / COSEM authentication tags to 96 bits, which prevents us from performing multiplication. Further research into feasibility of recovering the actual AAD may be warranted.

It is important to recall from section 3.1 that this capability is an intentional feature of GCM and does not violate its security claims. Although the way DLMS / COSEM uses its authentication key is insecure because it violates claim b in section 5.1 (the authentication key is required to produce an authentication tag,) GCM's integrity protection remains intact as long as the value of H (derived from the encryption key) remains secret.

Forgery Procedure

Let A_1 and A_2 be two AAD blocks, where A_1 contains either the byte 0x30 or 0x20, followed by the first 15 bytes of the unknown 15-bye (128-bit) authentication key and A_2 contains the last byte of the same key, followed by 15 bytes of padding. So we have two AAD blocks whose values we know a little about, but for the purposes of forging a MAC, we need not know anything about, except that they are fixed across multiple messages.

We are given one ciphered and authenticated message and the encryption key K with which it was encrypted. The authentication key remains unknown.

We start by computing H (section 3.4) and K_0 (section 3.2,) which both only require knowledge of K as well as X_n (section 3.4) which is not secret. Naturally, we know the ciphertexts C and C', such that $\lceil |C|/128 \rceil = \lceil |C'|/128 \rceil$, for the original and forged messages, respectively because we have the encryption key and were given C and its authentication tag T. We split the two messages into sets of 128-bit blocks $C_1, ..., C_n$ and $C'_1, ..., C'_n$ with appropriate padding.

Recall the equation for message authentication tag T from (section 3.4) and also that $GF(2^{128})$ is commutative and that since elements are polynomials over the finite field $\mathbb{Z}_2$, when two such polynomials $x = a_0 + a_1 x + ... + a_n x^n$ and $y = b_0 + b_1 x + \cdots + b_n x^n$, then it is always the case that $x + y = (a_0 \oplus b_0) x + (a_1 \oplus b_1) x^2 + \cdots + (a_n \oplus b_n) x^n$.

If we restrict ourselves to the addition operation then truncation followed by addition results in the same outcome as addition followed by truncation. Recall also that the addition and subtraction are the same operation over $\mathbb{Z}_2$.

The authentication tag T (before truncation – setting the coefficients of the 32 lowest degree terms to zero) is

$$T = A_1 H^4 + A_2 H^3 + C_1 H^2 + X_n H + K_0$$

and we make no distinction between truncated and full addition because the last 32 bits (coefficients) of T have no influence on the first 96 that we are interested in under addition, as shown above.

Since we know H, and X_n, we subtract the original message ciphertext $C_1 H^2$ and the counters $X_n H$ by computing

$$S = T + C_1 H^2 + X_n H = A_1 H^4 + A_2 H^3 + K_0$$

and have subtracted $X_n H$ because if we want to make the message a different length in bytes, but with the same number of blocks, a new value X'_n must be computed along with C'_1, which is the replacement ciphertext, padded to 128 bits.

Finally, we compute our falsified tag

$$T' = S + C'_1 H^2 + X'_n + K_0 = A_1 H^4 + A_2 H^3 + C'_1 H^2 + X'_n H + K_0$$

, truncate T' to 96 bits. Since only addition was used to manipulate T into T', only the unknown bits that we are not interested in will differ from a legitimate tag U that was

Encryption Key	6261636f6e20696e206d7920736f6170
Auth Key	69206d6164652069742064d7973656c66
IV	73696c6c7920726162626974
Original Plaintext	7765206b6e6f772063727970746f0000
New Plaintext	726561642074686520737065632100000
Original Ciphertext	1ced681e493abcdba500c7600a021291
New Ciphertext	19ed29110721a39ee601ce751d4c1291
Original MAC	58a5833ec6f4c259aeccced9
Expected MAC	ea20d182118dc78f995f2627

Table 7.1: Message Details for Forgery Attack

```
sage: from Crypto.Cipher import AES
sage: F.<x> = QQ[]
sage: K.<a> = GF( 2**128, name='a', modulus=x^128 + x^7 + x^2 + x + 1 )
sage: def bitreverse(i):
....:         return eval('0b' + bin(i)[2:].rjust(128,'0')[::-1])
....:
sage: c0     = K.fetch_int( bitreverse(0x7765206b6e6f772063727970746f0000) )
sage: c1     = K.fetch_int( bitreverse(0x726561642074686520737065632100000) )
sage: cipher = AES.new( "6261636f6e20696e206d7920736f6170".decode("hex"), AES.MODE_ECB )
sage: h      = eval('0x' + cipher.encrypt( ('0'*32).decode("hex") ).encode("hex"))
sage: H      = K.fetch_int( bitreverse( h ) )
sage: orig   = K.fetch_int( bitreverse(0x58a5833ec6f4c259aeccced900000000) )
sage: new    = hex(bitreverse( (orig + H^2 * c0 + H^2 * c1).integer_representation() ))[2:26]
sage: print(new)
ea20d182118dc78f995f2627
```

Figure 7.1: Computation of Forged MAC in SAGE

computed legitimately based on H, knowledge of A_1, A_2, C_1', X_n, and K_0

$$U = A_1H^4 + A_2H^3 + C_1'H^2 + X_n'H + K0$$

The attack works similarly when more than one block of ciphertext is present, except that more ciphertext blocks need to be subtracted and then added back.

Above, we assume knowledge of K because we need to encrypt a replacement message, but if our given authenticated message contains enough known plaintext, we can use the attack in section 7.3.3, and we then need only H. We tested this attack against an encrypted and authenticated messages created by taco section 6.9 using the SAGE open source mathematics program[8]. To reproduce our result, see the cryptographic details of the messages in table 7.1 and the SAGE session and code in fig. 7.1.

7.2 Implementation Vulnerabilities

This section is a list of implementation vulnerabilities that may come up in an implementation of DLMS / COSEM. While many of them did come up in our testing, others are only included for completeness.

7.2.1 Frame Counters Unenforced (Lack of Replay Protection)

A frame counter is considered to be unenforced if a device allows a ciphered APDU to be processed if the messages frame counter is less than or equal to the counter of the last enciphered message received from a device with the same title sending it.

This vulnerability can be easily checked by sending two messages with the same frame counter in succession.

7.2.2 Predictable Nonces

The values of nonces should not be predictable and nonces should be very unlikely to repeat. In order to test the randomness of nonces, a significant number of nonces (megabytes) must be collected and then subjected to statistical tests such as those suggested in[54].

7.2.3 Identical AA Titles Allowed

If identical titles are allowed, a client or server is willing to communicate with a DLMS / COSEM device that presents the same AP title as its own in its ACSE exchange. This issue is related the the HLS server impersonation attack from[12] described in section 7.1.3.

7.2.4 Ciphered APDU Types Ignored

The ciphered APDU type is said to be ignored if the inner message type of a ciphered APDU is not required to match the outer message type that describes the ciphered APDU.

The possibility does not indicate a problem with the protocol but if, for example, and intrusion detection system counts on the information always being accurate then it is important that the assumption is tested.

7.2.5 Cipher-Only APDUs Accepted

It is very unwise to allow cipher-only APDUs because they are highly susceptible to manipulation. No device should ever allow such messages, as is made clear by the attack described in section 7.3.3.

7.2.6 Degenerate Ciphered APDU's Accepted

We call an APDU that is packaged as a ciphered message, but that has both encryption and authentication disabled in its security control byte (described in section 5.4), with a plaintext message sent in place of an encrypted one. If such messages are accepted, a meter that has a policy only to respond to ciphered messages could be fooled into processing a message as plaintext, but authorizing the requested action as though it were ciphered.

7.2.7 Plaintext AARQ Elicits Encrypted AARE

This vulnerability exists if an AARQ message with a plaintext User-Information field results in a ciphered AARE message with an encrypted User-Information field. This behavior has been observed in lab devices and combined with vulnerabilities (7.2.5 and 7.2.3) and a default LLS password, was exploited as described in in attack 7.4.2.

7.2.8 Plaintext Authentication Allowed

If authentication (HLS or LLS) is allowed, then this vulnerability is present. LLS is completely vulnerable to passive interception because the password is sent in plaintext and proprietary, MD5, and SHA1 versions of HLS are vulnerable to MITM the attacks in section 7.1.3.

7.2.9 On-Line Dictionary Attack

A meter is vulnerable to on-line dictionary attacks if it is possible to verify whether a password is correct or incorrect more than a reasonable number of times per reasonable time period. During a period where the maximum number of failed access attempts has exceeded the number allowed in a time period, there should be no difference in behavior between the meters response upon presentation of a valid or invalid credential. The meter we tested temporarily disallowed login after a threshold number of failed login attempts but then gave a different error message when presented with a valid password than when presented with an invalid password. Timing may also be an indicator – for example: DSMR requires a meter to log certain events.

7.2.10 Message Authentication Not Enforced

This vulnerability exists if messages with invalid authentication tags are not rejected.

7.2.11 Non-HLS-GMAC Authentication Supported

Authentication protocols other than HLS GMAC (section 4.2.2) are supported. The DSMR lists HLS 3 and 4 as being enabled on all meters and that the GO cannot change that.

7.2.12 Arbitrary Client Titles Accepted

If a server (meter) allows a client with any title to connect then in order to operate correctly, it must track the last frame counter presented by every device it has ever communicated with. Since the device has finite memory and there are 2^{64} possible titles, a table can be filled by an attacker, resulting in denial of service. On the other hand, if the server uses e.g. a ring buffer, then an attacker may be able to connect with enough different names to overwrite the last frame counter of the client whose packets they wish to re-play. The only way to ensure that replay protection is possible is to configure the exact titles of clients that are allowed to connect to the meter.

7.3 Attack Building Blocks

The purpose of this section is to introduce some basic attacks building blocks that we can make use of in the compound attacks in section 7.4.

7.3.1 Acquiring Ciphertexts whose Plaintexts are Known

In the context of DLMS / COSEM, it is very likely that a great deal of information about the plaintext being sent is known by an attacker.

They must either guess, knowing that certain requests, like power utilization will inevitably be sent. While this information is helpful to an attacker, the following, which require different combinations of vulnerabilities to become exploitable, are also excellent sources of known plaintext:

- Ciphered User-Information fields from AARQ requests provide up to 15 bytes of known plaintext
- HLS MD5 provides 17 bytes if the password is unknown and a full 31 bytes if it is
- HLS SHA1 provides 17 bytes if the password is unknown, 37 if the password is known.
- HLS 5 (GMAC) authentication responses are by far the most reliable source of known plaintext that we have found. They provide 22 bytes of known plaintext with no knowledge of authentication credentials required.
  The first 17 bytes because it is HLS and an extra 6 bytes because the plaintext is composed of a ciphered (authenticate-only) APDU with another ciphered APDU inside it. The inner ciphered APDU has a 5 byte, predictable internal header:

$(0x10||counter-1)$. The internal header must be predictable because it is required that IVs of successive cryptograms increase by one. Thus, since the inner ciphered APDU must be encrypted before the one containing it, and not other APDUs will be sent between them, the counter of the inner message will be exactly one less than the counter on the outer one.

7.3.2 Authenticator Removal

The procedure for converting a valid encrypted and authenticated ciphered APDU (see figure 5.1) into a valid encrypted-only APDU is as follows:

- Remove the last 12 bytes of the APDU
- Subtract 12 from the length field
- Set the security control byte to 0x20 (indicating that there is no authentication tag present, see table 5.2)

7.3.3 Message Replacement

If *cipher-only APDUs are permitted* then we can use the recover the keystream of a validly encrypted message with known plaintext by by XOR'ing the known plaintext with the ciphertext as shown in section 3.3. Once the keystream is recovered, the keystream is applied to the replacement message of our choice. The only restriction is that the replacement message cannot exceed the length of the original message.

If the donor message was sent with an authentication tag then it must be removed as described in 7.3.2 because the manipulated ciphertext would probably be detected if paired with the authentication tag from the original message that we have now replaced.

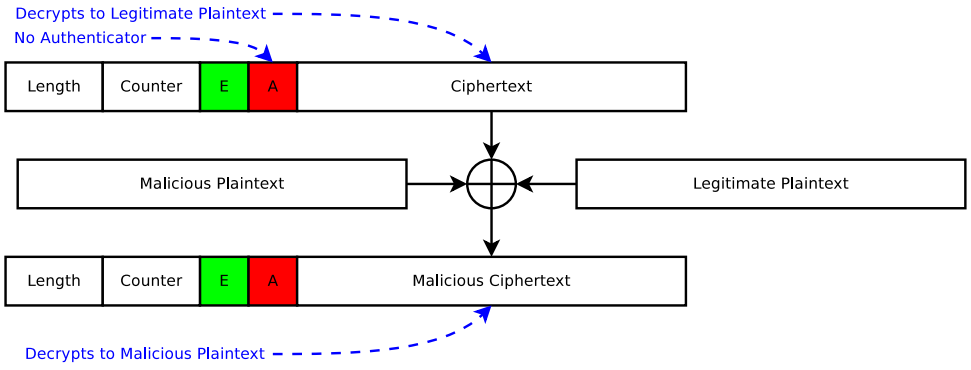

Figure 7.2: Message Replacement Attack

7.4 Compound Attacks

In this section, we describe several practical attacks / tests that have been developed specific to the DLMS / COSEM protocol, some of which requiring multiple simulta-

neous vulnerabilities. Control-flow (buffer overflow, stack overflow, format string, etc.) vulnerabilities that are specific to the hardware architecture of a specific meter have been excluded in favor of general protocol-level attacks that could apply to any DLMS / COSEM meter, regardless of hardware details.

The purpose of this section is to present the finding that multiple severe vulnerabilities not only can be present in one product, but there is at least one example where they in fact did. It is hoped that these findings will motivate stakeholders and system developers to integrate the attacks and vulnerability checks into their existing testing procedures.

Furthermore, it should be noted that the the attacks listed in this section were successfully executed against a pair of commercial smart meters[1] with security configured as prescribed by DSMR security requirements.

We were relieved to learn that the first set that we tested were operating on an out-dated firmware version. The same meter models with newer firmware were not vulnerable to the listed attacks.

7.4.1 Traffic Analysis

The primary purpose of DLMS / COSEM is for reading metering information, which involves a large number of read operations and a comparatively small number of write operations. The write operations that are likely to happen include but are not limited to clock setting, the schedule for tariff switches between e.g. night and day, connecting and disconnecting power, per KWh costs, and prepaid credit levels for pre-paid meters. An adversary controlling the connection between grid operator and meter will be able to tell whether a message is a Get or Set command and the size of the message, regardless of whether ciphering is enabled because the protocol requires it.

When a set command is issued, the APDU includes fixed size information that tells the meter which object is being set along with the variable size value to which the object is being set. This variability can leak information, particularly when the value being set is of an unusual size when compared to other values that are likely to be set.

One such unusually sized value is the manufacturer specific OBIS code that references an 8-bit value that in turn controls the state of the meter's power disconnection relay. Though we have not examined the representation of all settable data held in the meter, some have distinctive sizes and only one was found whose representation was an 8-bit integer, making it one of very few 15-byte APDUs.

If the size of a request can leak information about the object being accessed, the same applies to get responses from a meter. In this way, the presence of distinctive data structures in a ciphered APDU, whether a get or set, can be used to target undesirable messages (like power disconnection) and they can also be used to provide more detailed message

[1] very similar models from one manufacturer

information that improves an attackers ability to carry out the message replacement attacks described in 7.3.3

7.4.2 Ciphered Communication Without Keys

This exploit requires implementation vulnerabilities "Identical AA Titles Allowed" (section 7.2.3,) "Cipher-Only APDUs Accepted" (section 7.2.5,) "Plaintext AARQ Elicits Encrypted AARE" (section 7.2.7,) and knowledge of the LLS / HLS (MD5 and SHA1) authentication password (on our meters, they were the same password.) Replay protection may need to fail, depending on how it is implemented on the server.

When using LLS, replay is not required if the client maintains two lists of lowest allowed frame counters (one to track its own frame counter, and the others to track the frame counters of clients.) Please recall that the initialization vector for an enciphered DLMS / COSEM APDU is composed of the 64-bit AP title negotiated through ACSE, followed by a 32-bit frame counter that is attached to the message being delivered. Recall also, that the User-Information field of ACSE can be either encrypted or not, and when it is, the enciphering is exactly the same as any other ciphered DLMS / COSEM APDU, down to the increasing frame counter and that it can be transferred either in plaintext or encrypted form.

If the meter allows a client presenting an AP title identical to its own to connect[2] but otherwise enforces a monotone increasing frame counter (and the split-brain scenario described above holds,) then an enciphered message from the server has a frame counter, key, and thus IV and keystream that meets all other requirements. If an attacker knows, or can manipulate the plaintext of a message (such as an encrypted User-Information field,) then then that attacker now has everything required to construct a valid enciphered DLMS / COSEM APDU that will be accepted by the server.

The legitimate client implementation used for the research sends a user information field is encoded in 14 bytes and does not make use of the optional proposed quality of service field, which is also not in use in the protocol version in play. The AARQ user information field (xDLMS-Initiate.request), sent in plaintext (along with an accepting password – the default one,) was responded to with an enciphered (and authenticated) xDLMS-Initiate.request message that also mentioned nothing of the proposed quality of service and so also happened to be 14 bytes long. When presented with a slightly modified, 15 byte plaintext xDLMS-Initiate.request message, this time with a proposed quality of service. The result was a 15-byte xDLMS-Initiate.response message and the message decrypted the same every time.

The requests and responses were as follows (in plaintext):

[2]the AP title of the meter is obtainable by various methods, like sending the first message in an HLS session and recording the AP title presented in the AARE message

Request	Response
01000000..065f1f0400007e1fffff	0800..065f1f0400007e1f04300007
0100000100065f1f0400007e1fffff	080100065f1f0400007e1f04300007

At this point, we have a ciphered APDU provided by the server that is acceptable to the server, and whose plaintext we can replace with the 15-byte or smaller message of our choosing, as described in 7.3.3. Naturally, the first choice is a set command that causes power disconnection and a satisfying "click," followed by loss of power to a computer that had been hooked to the meter so that it would register some power utilization.

While we can power off the meter using this attack, if we had one more byte of known plaintext, we could save ourselves some trouble and just go ahead and disable encryption.

While a proof of concept has not been implemented, the attack is very likely iterable – a get command is only 13 bytes long and we can use that command to retrieve any large data structure whose value is known ahead of time. For example, we might use the ASCII serial number representation mentioned in 7.1.5, thus trading 13 bytes of keystream for a fresh (increased frame counter) ciphered APDU with a larger number of known plaintext bytes that could be used to simply turn off encryption, and thus enabling attacks on confidentiality.

7.4.3 Command Injection Proxy

The attack depends upon "Frame Counters Unenforced" (section 7.2.1,) "Cipher-Only APDUs Accepted" (section 7.2.5,) and uses HLS GMAC for known plaintext.

A command injection proxy, similar to the one constructed in the research portion of this work, can be constructed. The principles are very similar to the attack in 7.4.2, but instead of performing what is effectively an MITM attack between a server and itself, we place our proxy between a legitimate client and a legitimate server. The proxy can be made to work with other authentication methods but we use HLS-5 because it maximizes keystream and irony simultaneously – 21 bytes of known plaintext for 5, as opposed to only 17 for 2[3], 3, and 4.

A proxy is implemented that upon connection from a client, establishes a connection to the legitimate server. Every message received from the client is sent unchanged to the server and every message from the server is sent unchanged to the client, except that when the client sends it's HLS response message, it's plaintext is computed as described in 7.3.3, which is then used to recover the 21 bytes of keystream, which are then stored along with the title and sequence number of the client's HLS response message for later use.

After the keystream, title, and frame counter are stored, the proxy continues to allow

[3]knowledge of the hash function is not required for this attack

communication to continue unchanged. When the client disconnected, the proxy then injects whatever commands are desired.

The proof of concept also happened to power off the meter at the end of the session, but with 21 bytes of keystream available, there are many other options.

A more general approach would be to send the 16-byte command to disable encryption and authentication, followed by a command to change one of the LLS / HLS-3/HLS-4 passwords to say, "taco." At that point, we could read/write anything we like, except the ciphering key and then turn encryption back on. Since the legitimate user is only allowed to use the more secure HLS-5 authentication mechanism that does not use the password we set, the grid operator will most likely be able to connect to the meter with no problems.

Chapter 8

Conclusion

We have presented the basic approach to vulnerability analysis in chapter 2 and validated it through a limited but fruitful application to the DLMS / COSEM protocol, verified implementation vulnerabilities in one commercial product and verified that the vulnerability had been corrected in a later release of the same product.

In section section 7.1, we have identified some surprisingly elementary design flaws in the DLMS / COSEM protocol itself, including completely avoidable leaking of message type information and a textbook replay and dictionary attacks against plaintext HLS (section 7.1.3 and 7.1.4.)

Cryptographically speaking, (a) the protocol has optional message authentication that is indicated insecurely (section 7.1.1,) which can lead to the ability to disable message authentication and thus replace the plaintext of a message without detection and (b) makes us of its message authentication key in such a way that it can be bypassed in certain common situations (section 7.1.6) with only knowledge of the encryption key and in a manner described in the specification of it's stream cipher GCM.

A set of generic laboratory tools were developed (chapter 6), including dissectors supporting the most common parts of the protocol suite, cryptographic tools, and exploits that combine multiple vulnerabilities. We have also shown that it is indeed possible to have multiple simultaneous vulnerabilities present in a single meter by successfully completing compound attacks against two popular commercial smart meters.

8.1 Vulnerability Overview

A fixed exploit facilitates the repeating of tests against multiple smart meters, particularly in the case of compound attacks that are by their nature more difficult to carry out manually. Table 8.1 summarizes the vulnerabilities that we have discussed and whether they have been used to successfully attack an actual meter.

Section	Description	Exploited	Tool
7.1.1	Separable Authentication Ciphering	yes	7.4.3, 7.4.2
7.1.2	Message Type Leakage	-	-
7.1.3	HLS Server Impersonation	-	-
7.1.4	HLS Off-Line Dictionary Attack	-	-
7.1.5	Response Not Tied to Requests	-	-
7.1.6	MAC Forgery Without Authentication Key	-	-
7.2.1	Frame Counters Unenforced	yes	7.4.3
7.2.2	Predictable Nonces	-	-
7.2.3	Identical AA Titles Allowed	yes	7.4.2
7.2.4	Ciphered APDU Types Ignored	-	-
7.2.5	Cipher-Only APDU's Accepted	yes	7.4.3, 7.4.2
7.2.6	Degenerate Ciphered APDU's Accepted	-	-
7.2.7	Plaintext AARQ Elicits Ciphered AARE	yes	7.4.2
7.2.8	Plaintext Authentication Allowed	yes	7.4.2
7.2.9	On-Line Dictionary Attack	yes	python script
7.2.10	Message Authentication not Enforced	-	-
7.2.11	Non-GMAC HLS Supported	-	-
7.2.12	Arbitrary Client Titles Accepted	yes	7.4.2
7.3.1	Acquiring Ciphertexts With Known Plaintext	yes	7.4.3, 7.4.2
7.3.2	Authenticator Removal	yes	7.4.3, 7.4.2
7.3.3	Message Replacement	yes	7.4.3, 7.4.2

Table 8.1: Exploited Vulnerabilities

MAC Forgery Without Authentication Key is a protocol vulnerability and was discovered after the practical portion of the project had ended and was tested against the DLMS / COSEM ciphering implementation in taco. The mathematical argumentation supporting the attack can be found in section 7.1.6 and the details required to independently reproduce the example attack are included in table 7.1 (cryptography details) and fig. 7.1 (output of a sage session computing the forgery.)

8.2 Recommendations

8.2.1 Private APN Issues

The security issues presented in chapter 7, particularly "Ciphered Communication Without Keys" (section 7.4.2,) could help an attacker gain the ability to send arbitrary commands to vulnerable meters in a cyber attack provided the attacker has access to the data communication network or a grid operator network.

This threat can be mitigated by only allowing field devices to communicate with the AMI HE system. After the mitigation an attacker can only exploit a vulnerable meter using the P3 interface if the attacker also has access to the AMI HE system, thus substantially

reducing the likelihood of successful exploitation.

8.2.2 Open Security Protocols

Regardless of whether the vulnerabilities we present in chapter 7 are exploitable in any given use case, their presence strongly suggests that the DLMS User Association would do well to embrace their core metrology competencies and adopt an existing, mature transport layer security protocol like Transport Layer Security (TLS) 1.2[57] rather than continuing to tack new features on to their custom protocol.

At minimum, the security protocol development process should be opened to the general public so that vulnerabilities in future protocol versions will be found before the protocol is put into production rather than after.

TLS 1.2 is recommended for the following reasons:

- TLS 1.2 adds secure mutual authentication using pre-shared keys – no certificates required.
- The pre-shard key mutual authentication in TLS can be keyed the existing DLMS / COSEM message and HLS-GMAC authentication key.
- TLS has a minimum set of algorithms required to be present in a compatible implementation and the DLMS / COSEM standard can add their own minimum requirements to ensure compatibility across vendors.
- TLS is used in many high profile systems and thus benefits from the intense scrutiny of security experts around the world. Furthermore, secure, high-performance implementations like OpenSSL[7] are freely available and under licensing terms that allow them to be used in proprietary software.

Since TLS is only specified for use over TCP and UDP, the submission of an informational Request for Comments (RFC) describing its use over HDLC will help ensure that implementations are inter-operable.

8.2.3 Smart Metering Gateway

Asset owners and regulators may consider adopting the gateway for a smart metering systems described in [44]. The gateway provides a pluggable module that mediates the needs of all smart metering stakeholders and protects smart meters from direct attack.

Appendix A

Wireshark Dissector Fields

Identifier	Description
wpdu.version	Version
wpdu.src	Source Port
wpdu.dst	Dest Port
wpdu.len	APDU Length
wpdu.apdu	APDU

Table A.1: Wrapper PDU Dissector Fields

Identifier	Description
hdlc.format	Format
hdlc.format.type	Type
hdlc.format.fragment	Fragment
hdlc.format.length	Length
hdlc.format.validlen	Valid Length
hdlc.src	Source Address
hdlc.dst	Destination Address
hdlc.control	Control
hdlc.control.type	Frame Type
hdlc.type.info	Information
hdlc.type.super	Supervisory
hdlc.super.cmd	Function
hdlc.unnum	Unnumbered
hdlc.unnum.c1	c1
hdlc.unnum.c2	c2
hdlc.unnum.cmd	Command
hdlc.llc	Logical Link Control
hdlc.llc.src	Source LSAP
hdlc.llc.dst	Destination LSAP
hdlc.llc.quality	Quality
hdlc.llc.info	LLC Information
hdlc.hcs	Header Check Sequence
hdlc.hcs	Frame Check Sequence
hdlc.rxseq	TX Sequence Number
hdlc.txseq	RX Sequence Number
hdlc.pf	Poll/Final

Table A.2: HDLC Dissector Fields

Identifier	Description
cosem.capdu	Ciphered APDU
cosem.capdu.len	Length
cosem.capdu.sc	Security Control
cosem.capdu.enc	Encrypted
cosem.capdu.auth	Authenticated
cosem.capdu.secsuite	Security Suite
cosem.capdu.keyset	Key Set
cosem.capdu.counter	Frame Counter
cosem.capdu.ciphertext	Ciphertext
cosem.capdu.auth	Auth Tag
cosem.capdu.title	Title
cosem.capdu.iv	IV
cosem	APDU
cosem.tag	PDU Tag
cosem.pdulen	PDU Length
cosem.tag.data	Data
cosem.aarq.tag	AARQ Tag
cosem.aarq.tag	Data
cosem.method	Method Descriptor
cosem.method.classid	Class ID
cosem.method.obis	OBIS Code
cosem.method.attribute	Attribute ID
cosem.data	Data
cosem.data.octetstring	Octet String
cosem.data.uint8	8-Bit Unsigned
cosem.data.uint16	16-Bit Unsigned
cosem.data.uint32	32-Bit Unsigned
cosem.invoke	Invocation Parameters
cosem.invoke.id	Identifier
cosem.invoke.class	Service Class
cosem.invoke.priority	Priority
cosem.actreq	Action Request
cosem.actreq.type	Type
cosem.actreq.params	Parameters
cosem.actrest	Action Response
cosem.actresp.type	Type
cosem.actresp.result	Result
cosem.actresp.return	Return Parameters
cosem.getresp	Get Response
cosem.getresp.type	Type
cosem.getresp.result	Result
cosem.getreq	Get Request
cosem.getreq.type	Type

Table A.3: COSEM Dissector Fields

Bibliography

[1] DLMS User Association Web Site. http://www.dlms.com.

[2] Future Technology Devices International Ltd. http://www.ftdichip.com.

[3] GuruX Project. http://www.gurux.fi.

[4] Lua Section of the Wireshark Wiki. http://wiki.wireshark.org/Lua.

[5] OpenMUC JDLMS Project. http://www.openmuc.org/index.php?id=42.

[6] OpenMUC Project. http://www.openmuc.org.

[7] OpenSSL Project. http://www.openssl.org .

[8] Sage: Open Source Mathematics Software. http://www.sagemath.org.

[9] Short name or logical name addressing? *http://www.dlms.com/ faqanswers/ questionsonthedlmscosem/ specification/ shortnameorlogicalnamereferencing.php*. DLMS User Association FAQ.

[10] Termineter Project. http://code.google.com/p/termineter .

[11] Wireshark Web Site. http://www.wireshark.org.

[12] Work Package 2.3 Research Smart Grid communication protocols and infrastructures to incorporate data security measures. Technical report, Expert Group on the security and resilience of Communication networks and Information systems for Smart Grids, 15 March 2012.

[13] The Dark Side of the Smart Grid SMart Meters (in)Security. Technical report, C4 Security, 2009.

[14] Functional Reference Architecture for Communications in Smart Metering Systems. Technical report, CEN, CENELEC, ETSI, 2011.

[15] CEN - CENELEC - ETSI Smart Grid Coordination Group Smart Grid Reference Architecture. Technical report, CEN,CENELEC,ETSI, 2012.

[16] DLMS/COSEM Information Security. *http://dlms.com/DLMSimages/Security.pdf*, Oct 2012. date from PDF metadata.

[17] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFCs 4634, 6234.

[18] Ross Anderson and Shailendra Fuloria. Who controls the off switch? In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 96–101. IEEE, 2010.

[19] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in CryptologyCRYPTO94*, pages 216–233. Springer, 1994.

[20] C Bennett, B Brown, B Singletary, D Highfill, D Houseman, F Cleveland, H Lipson, J Ivers, J Gooding, J McDonald, et al. Ami system security requirements. *Utility Commun. Arch. Int. User Grp.(UCAIUG), Raleigh, NC, USA, UCAIUG: AMI SEC ASAP*, 2008.

[21] Companion Specification for Energy Metering – DLMS/COSEM Architecture and Protocols. Norm, DLMS User Association, December 2009.

[22] Companion Specification for Energy Metering – DLMS/COSEM Architecture and Protocols. Norm, DLMS User Association, December 2009.

[23] GPRS Requirements, Dutch Smart Meter Requirements v4.0 Final. Norm, Netbeheer Nederland, Den Haag, The Netherlands, April 2011.

[24] P1 Companion Standard, Dutch Smart Meter Requirements v4.0 Final. Norm, Netbeheer Nederland, Den Haag, The Netherlands, April 2011.

[25] P2 Companion Standard, Dutch Smart Meter Requirements v4.0.5 Final. Norm, Netbeheer Nederland, Den Haag, The Netherlands, May 2012.

[26] P3 Companion Standard, Dutch Smart Meter Requirements v4.0 Final. Norm, Netbeheer Nederland, Den Haag, The Netherlands, April 2011.

[27] Dutch Smart Meter Requirements v4.0 Final. Norm, Netbeheer Nederland, Den Haag, The Netherlands, April 2011.

[28] Expert Group 1. Functionalities of smart grids and smart meters. Technical report, EU Commission Task Force for Smart Grids, 2010.

[29] Niels Ferguson. Authentication weaknesses in gcm. *Comments submitted to NIST Modes of Operation Process*, 2005.

[30] Russell Housley. Using advanced encryption standard (aes) counter mode with ipsec encapsulating security payload (esp). 2004.

[31] icube Software. DLMS Security Basics. http://www.cyamon.com/ Security/ security1.html.

[32] Distribution automation using distribution line carrier systems – Part 6: A-XDR encoding rule, 2000.

[33] Electricity metering data exchange The DLMS/COSEM suite Part 21: Direct local data exchange, 2002.

[34] Electricity metering - Data exchange for meter reading, tariff and load control - Part 46: Data link layer using HDLC protocol, 2007.

[35] Electricity metering - Data exchange for meter reading, tariff and load control - Part 47: COSEM transport layers for IPv4 networks, 2007.

[36] Electricity metering data exchange – The DLMS/COSEM suite – Part 5-3: DLMS/COSEM application layer, 2013.

[37] Electricity metering Data exchange for meter reading, tariff and load control Part 53: COSEM application layer, 2006.

[38] Electricity metering data exchange The DLMS/COSEM suite Part 61, 2007.

[39] Information technology – ASN. 1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 2002.

[40] InGuardians, Inc. Advanced Metering Infrastructure Attack Methodology. 2009.

[41] Joux, Antoine. Authentication failures in NIST version of GCM. *NIST Comment*, 2006.

[42] Sander Keemink and Bard Roos. Security Analysis of Dutch Smart Metering Systems. Technical report, University of Amsterdam, Jul 2008.

[43] Gyzy Kmethy. IEC 62056 DLMS/COSEM Workshop Part 2: Overview of main concepts. *http://www.dlms.com/training/TPAK2_DLMStraining_Ams2013_GKVV131008.pdf*, Oct 2013.

[44] Helge Kreutzmann, Stefan Vollmer, Nils Tekampe, and Arnold Abromeit. Protection profile for the gateway of a smart metering system (gateway pp). *Federal Office for Information Security Germany, Protection Profile*, 1(01), 2011.

[45] Kunal Adak, Jawash Mohamed, Sri Haritha Darapuneni. ADVANCED METERING INFRASTRUCTURE SECURITY. Technical report, University of Colorado, Boulder, 2009.

[46] John Larmouth. *ASN. 1 complete*. Morgan Kaufmann, 2000.

[47] David McGrew and John Viega. The Galois/Counter mode of operation (GCM). *Submission to NIST*, 2004.

[48] Basisfuncties voor de meentinrichting voor elektriciteit, gas, en thermische energie voor kleinverbruikers. NEN, Delft, Nederland, August 2007.

[49] European Parliament and Council of the European Union. Directive 2004/22/EC, 2010.

[50] Philippe Oechslin. OPHCRACK (the time - memory - trade - off - cracker. http://lasecwww.epfl.ch/ oechslin/projects/ophcrack/.

[51] Venkat Pothamsetty and Bora A Akyol. A vulnerability taxonomy for network protocols: Corresponding engineering best practice countermeasures. In *Communications, Internet, and Information Technology*, pages 168–175, 2004.

[52] Gordon Procter and Carlos Cid. On weak keys and forgery attacks against polynomial-based mac schemes. *IACR Cryptology ePrint Archive*, 2013:144, 2013.

[53] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992. Updated by RFC 6151.

[54] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.

[55] Markku-Juhani O Saarinen. Gcm, ghash and weak keys. *IACR Cryptology ePrint Archive*, 2011:202, 2011.

[56] Markku-Juhani O Saarinen. Sgcm: The sophie germain counter mode. *IACR Cryptology ePrint Archive*, 2011:326, 2011.

[57] T. Dierks. The Transport Layer Security (TLS) Protcol Version 1.2. RFC 5246, 2008.

[58] International Telegraph and Telephone Consultative Committee. *CCITT Recommendation X.227: Data Communication Networks : Connection-oriented Protocol Specification for the Association Control Service Element*. International Telecommunication Union, 1992.

[59] John Viega and David A McGrew. The use of galois/counter mode (gcm) in ipsec encapsulating security payload (esp). 2005.

[60] Vinoo S. Warrier. DLMS/COSEM Security Enhancement - Latest Updates. *http://www.kalkitech.com/articles/DLMS%20COSEM%20Security%20Enhancement-Latest%20Updates.pdf*, May 2011. date from PDF metadata.

[61] Wikipedia. High-Level Data Link Control. [Online; accesed 4-March-2014].

[62] Wikipedia. Purdue Enterprise Reference Model — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 2-March-2014].