13/1851/CDV

COMMITTEE DRAFT FOR VOTE (CDV)

PROJECT NUMBER: **IEC 62056-5-3 ED4**	
DATE OF CIRCULATION: **2021-12-03**	CLOSING DATE FOR VOTING: **2022-02-25**
SUPERSEDES DOCUMENTS: **13/1824A/RR**	

IEC TC 13 : ELECTRICAL ENERGY MEASUREMENT AND CONTROL	
SECRETARIAT: Hungary	SECRETARY: Mr Bela Bodi
OF INTEREST TO THE FOLLOWING COMMITTEES: TC 57	PROPOSED HORIZONTAL STANDARD: ☐ Other TC/SCs are requested to indicate their interest, if any, in this CDV to the secretary.
FUNCTIONS CONCERNED: ☐ EMC ☐ ENVIRONMENT	☐ QUALITY ASSURANCE ☐ SAFETY
☒ SUBMITTED FOR CENELEC PARALLEL VOTING **Attention IEC-CENELEC parallel voting** The attention of IEC National Committees, members of CENELEC, is drawn to the fact that this Committee Draft for Vote (CDV) is submitted for parallel voting. The CENELEC members are invited to vote through the CENELEC online voting system.	☐ NOT SUBMITTED FOR CENELEC PARALLEL VOTING

This document is still under study and subject to change. It should not be used for reference purposes.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

TITLE:

Electrcity metering data exchange - The DLMS/COSEM suite - Part 5-3: DLMS/COSEM application layer

PROPOSED STABILITY DATE: 2024

NOTE FROM TC/SC OFFICERS:

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE –

Part 5-3: DLMS/COSEM application layer

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this International Standard may involve the use of a maintenance service concerning the stack of protocols on which the present standard IEC 62056-5-3 is based.

The IEC takes no position concerning the evidence, validity and scope of this maintenance service.

The provider of the maintenance service has assured the IEC that he is willing to provide services under reasonable and non-discriminatory terms and conditions for applicants throughout the world. In this respect, the statement of the provider of the maintenance service is registered with the IEC. Information may be obtained from:

DLMS[1] User Association
Zug/Switzerland
www.dlms.com

____

1 Device Language Message Specification.

International Standard IEC 62056-5-3 has been prepared by IEC technical committee 13: Electrical energy measurement and control.

This third edition cancels and replaces the second edition of IEC 62056-5-3, published in 2016. It constitutes a technical revision.

The significant technical changes with respect to the previous edition are listed in Annex K (Informative).

The text of this standard is based on the following documents:

FDIS	Report on voting
13/XX/FDIS	13/XX/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all the parts in the IEC 62056 series, published under the general title *Electricity metering data exchange– The DLMS®/COSEM suite*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC website under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

The National Committees are requested to note that for this publication the stability date is 2021.

THIS TEXT IS INCLUDED FOR THE INFORMATION OF THE NATIONAL COMMITTEES AND WILL BE DELETED AT THE PUBLICATION STAGE.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

This fourth edition of IEC 62056-5-3 has been prepared by IEC TC13 WG14 with a significant contribution of the DLMS® User Association, its A-type liaison partner.

This edition is in line with DLMS® UA 1000-2, the “Green Book” Ed. 10:2020 and DLMS® UA 1000-2, the “Green Book” Ed. 10 Amendment 1 2021 .

ELECTRICITY METERING DATA EXCHANGE – THE DLMS®/COSEM SUITE –

Part 5-3: DLMS®/COSEM application layer

1 Scope

This part of IEC 62056 specifies the DLMS®/COSEM application layer in terms of structure, services and protocols for DLMS®/COSEM clients and servers, and defines rules to specify the DLMS®/COSEM communication profiles.

It defines services for establishing and releasing application associations, and data communication services for accessing the methods and attributes of COSEM interface objects, defined in IEC 62056-6-2:2021 using either logical name (LN) or short name (SN) referencing.

Annex A (normative) defines how to use the COSEM application layer in various communication profiles. It specifies how various communication profiles can be constructed for exchanging data with metering equipment using the COSEM interface model, and what are the necessary elements to specify in each communication profile. The actual, media-specific communication profiles are specified in separate parts of the IEC 62056 series.

Annex B (normative) specifies the SMS short wrapper.

Annex C (normative) specifies the gateway protocol.

Annex D, Annex E and Annex F (informative) include encoding examples for APDUs.

Annex G (normative) provides NSA Suite B elliptic curves and domain parameters.

Annex H (informative) provides an example of an End entity signature certificate using P-256 signed with P-256.

Annex I (normative) specifies the use of key agreement schemes in DLMS®/COSEM.

Annex J (informative) provides examples of exchanging protected xDLMS APDUs between a third party and a server.

Annex K (informative) lists the main technical changes in this edition of the standard.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61334-4-41:1996, *Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 41: Application protocols – Distribution line message specification*

IEC 61334-6:2000, *Distribution automation using distribution line carrier systems – Part 6: A-XDR encoding rule*

IEC TR 62051:1999, *Electricity metering – Glossary of terms*

IEC TR 62051-1:2004, *Electricity metering – Data exchange for meter reading, tariff and load control – Glossary of terms – Part 1: Terms related to data exchange with metering equipment using DLMS®/COSEM*

IEC 62056-6-2:2021, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 6-2: COSEM interface classes*

IEC 62056-8-3:2013, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 8-3: Communication profile for PLC S-FSK neighbourhood networks*

ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO/IEC 8825-1:2008, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

ISO/IEC 15953:1999, *Information technology – Open Systems Interconnection – Service definition for the Application Service Object Association Control Service Element*

NOTE This standard cancels and replaces ISO/IEC 8649:1996 and its Amd. 1:1997 and Amd. 2:1998, of which it constitutes a technical revision.

ISO/IEC 15954:1999, *Information technology – Open Systems Interconnection – Connection-mode protocol for the Application Service Object Association Control Service Element*

NOTE This standard cancels and replaces ISO/IEC 8650-1:1999 and its Amd. 1:1997 and Amd. 2:1998, of which it constitutes a technical revision.

3 Terms, definitions, abbreviations and symbols

For the purposes of this document, the terms and definitions given in IEC TR 62051:1999, in IEC TR 62051-1, in RFC 4106, and the following apply.

3.1 General DLMS®/COSEM definitions

3.1.1
ACSE APDU
APDU used by the Association Control Service Element (ACSE)

3.1.2
application association
cooperative relationship between two application entities, formed by their exchange of application protocol control information through their use of presentation services

3.1.3
application context
set of application service elements, related options and any other information necessary for the interworking of application entities in an application association

3.1.4
application entity
system-independent application activities that are made available as application services to the application agent, e.g., a set of application service elements that together perform all or part of the communication aspects of an application process

3.1.5
application process
an element within a real open system which performs the information processing for a particular application

[SOURCE: ISO/IEC 7498-1:1994, 4.1.4]

3.1.6
authentication mechanism
the specification of a specific set of authentication-function rules for defining, processing, and transferring authentication-values

[SOURCE: ISO/IEC 15953:1999, 3.5.11]

3.1.7
block

one portion of an xDLMS APDU; the payload of a GBT APDU

3.1.8
client
application process running in the data collection system

3.1.9
client/server
relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfils the request

3.1.10
confirmed GBT procedure
procedure in which the sender sends streams of GBT APDUs and at the end of each stream the recipient acknowledges the blocks received and attempts recovering any missing blocks

Note 1 to entry: A GBT stream consists of one or more GBT APDUs.

Note 2 to entry: In the case of a confirmed GBT stream the end of the stream is indicated by the streaming bit to set to FALSE (0). In the case of an unconfirmed GBT stream the end of the stream is indicated by the Final bit set to TRUE (1).

3.1.11
COSEM
Companion Specification for Energy Metering; refers to the COSEM object model

3.1.12
COSEM APDU
comprises ACSE APDUs and xDLMS APDUs

3.1.13
COSEM data
COSEM object attribute values, method invocation and return parameters

3.1.14
COSEM interface class
entity with specific set of attributes and methods modelling a certain function on its own or in relation with other COSEM interface classes

3.1.15
COSEM object
instance of a COSEM interface class

3.1.16
DLMS®/COSEM
refers to the application layer providing xDLMS services to access COSEM interface object attributes. Also refers to the DLMS®/COSEM Application layer and the COSEM data model together.

3.1.17
DLMS® context
a specification of the service elements of DLMS® and semantics of communication to be used during the lifetime of an application association

[SOURCE: IEC 61334-4-41:1996, 3.3.5]

3.1.18
entity authentication
corroboration that an entity is the one claimed

[SOURCE: ISO/IEC 9798-1:2010, 3.14]

3.1.19
gap
empty space i.e. missing blocks in the receive queue RQ

Note to entry: A receive queue RQ may have one or more gaps. In each gap, one or more blocks may be missing.

3.1.20
GBT APDU
xDLMS APDU with control information that carries a block of another xDLMS APDU or an empty block

3.1.21
GBT exchange
exchanging GBT APDUs that carry the service primitives of the same service

3.1.22
GBT stream
a sequence of GBT APDUs

3.1.23
general block transfer
GBT
DLMS®/COSEM application layer mechanism that can transfer any other xDLMS APDU that would be otherwise too long to fit into the maximum APDU size negotiated, in several blocks.

Note to entry: GBT can be forced by including GBT parameters in the .request service primitive.

3.1.24
logical device
abstract entity within a physical device, representing a subset of the functionality modelled with COSEM objects

3.1.25
master
central station – station which takes the initiative and controls the data flow

3.1.26
message

xDLMS APDU carrying a service primitive in an encoded form, which may also be cryptographically protected

3.1.27
mutual authentication
entity authentication which provides both entities with assurance of each other's identity

Note 1 to entry: The DLMS®/COSEM HLS authentication mechanism provides mutual authentication.

[SOURCE: ISO/IEC 9798-1:2010, 3.18, modified by adding Note 1]

3.1.28
overflow
more GBT APDUs received in one stream than the size of the GBT window

3.1.29
physical device
physical metering equipment, the highest level element used in the COSEM interface model of metering equipment

3.1.30
pull operation
style of communication where the request for a given transaction is initiated by the client

3.1.31
push operation
style of communication where the request for a given transaction is initiated by the server

3.1.32
receive queue
RQ
placeholder for the blocks of the APDU received in a GBT stream

3.1.33
server
an application process running in a metering equipment

3.1.34
send queue
SQ
placeholder for the blocks of the APDU to be sent

3.1.35
service-specific block transfer
DLMS®/COSEM application layer mechanism that can transfer an xDLMS APDU corresponding to a specific service primitive, that would be otherwise too long to fit into the maximum APDU size negotiated, in several blocks

3.1.36
streaming window
number of GBT APDUs that can be received in a stream

3.1.37
slave
station responding to requests of a master station.

Note to entry: A meter is normally a slave station.

3.1.38
system title
unique identifier of the system

3.1.39
unconfirmed GBT procedure
procedure in which the sender sends and the recipient receives a single stream of GBT APDUs, the recipient does not acknowledge the blocks received and does not attempt to recover any blocks lost

Note to entry: This is used to carry unconfirmed service requests from the client to the server or unsolicited service requests from the server to the client.

3.1.40
unilateral authentication
entity authentication which provides one entity with assurance of the other's identity but not vice versa

Note 1 to entry: The DLMS®/COSEM LLS authentication mechanism provides unilateral authentication.

[SOURCE: ISO/IEC 9798-1:2010, 3.39]

3.1.41
xDLMS
extended DLMS®; refers to the DLMS® protocol with the extensions specified in this standard

3.1.42
xDLMS APDU
APDU used by the xDLMS Application Service Element (xDLMS ASE)

3.1.43
xDLMS message
xDLMS APDU exchanged between a client and a server or between a third party and a server

3.2 Definitions related to cryptographic security

3.2.1
access control
restricts access to resources to only privileged entities

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.2
asymmetric key algorithm
see Public key cryptographic algorithm

3.2.3
authentication
a process that establishes the source of information, provides assurance of an entity's identity or provides assurance of the integrity of communications sessions, messages, documents or stored data

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.4
authentication code
a cryptographic checksum based on an approved security function (also known as a Message Authentication Code)

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.5
certificate
see public key certificate

3.2.6
Certification Authority
CA
the entity in a Public Key Infrastructure (PKI) that is responsible for issuing public key certificates and exacting compliance to a PKI policy

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.7
Certificate Policy
CP
a specialized form of administrative policy tuned to electronic transactions performed during certificate management. A Certificate Policy addresses all aspects associated with the generation, production, distribution, accounting, compromise recovery, and administration of digital certificates. Indirectly, a certificate policy can also govern the transactions conducted using a communications system protected by a certificate-based security system. By controlling critical certificate extensions, such policies and associated enforcement technology can support provision of the security services required by particular applications.

[SOURCE: NIST SP 800-32:2001]

3.2.8
challenge
a time variant parameter generated by a verifier

[SOURCE: ITU-T X.811:1995, 3.8]

3.2.9
ciphering
authentication and / or encryption using symmetric key algorithms

3.2.10
ciphertext
data in its encrypted form

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.11
cofactor
the order of the elliptic curve group divided by the (prime) order of the generator point (i.e. the base point) specified in the domain parameters

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.12
confidentiality
the property that sensitive information is not disclosed to unauthorized entities

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.13
cryptographic algorithm
a well-defined computational procedure that takes variable inputs including a cryptographic key and produces an output

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.14
cryptographic key
key
a parameter used in conjunction with a cryptographic algorithm that determines its operation in such a way that an entity with knowledge of the key can reproduce or reverse the operation, while an entity without knowledge of the key cannot

Note 1 to entry:

Examples include:

1. The transformation of plaintext data into ciphertext data,
2. The transformation of ciphertext data into plaintext data,
3. The computation of a digital signature from data,
4. The verification of a digital signature,
5. The computation of an authentication code from data,
6. The verification of an authentication code from data and a received authentication code,
7. The computation of a shared secret that is used to derive keying material.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.15
cryptoperiod
the time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.16
dedicated key
in DLMS®/COSEM, a symmetric key used within a single instance of an Application Association. See also session key

3.2.17
deprecated
not recommended for new implementations

3.2.18
digital signature
the result of a cryptographic transformation of data that, when properly implemented with supporting infrastructure and policy, provides the services of:

1) origin authentication
2) data integrity, and
3) signer non-repudiation

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.19
directly trusted CA
a directly trusted CA is a CA whose public key has been obtained and is being stored by an end entity in a secure, trusted manner, and whose public key is accepted by that end entity in the context of one or more applications

[SOURCE: ISO/IEC 15945:2002, 3.4]

3.2.20
directly trusted CA key
a directly trusted CA key is a public key of a directly trusted CA. It has been obtained and is being stored by an end entity in a secure, trusted manner. It is used to verify certificates without being itself verified by means of a certificate created by another CA.

Note 1 to entry: Directly trusted CAs and directly trusted CA keys may vary from entity to entity. An entity may regard several CAs as directly trusted CAs.

[SOURCE: ISO/IEC 15945:2002, 3.5]

3.2.21
distribution
see key distribution

3.2.22
domain parameters
the parameters used with a cryptographic algorithm that are common to a domain of users

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.23
encryption
the process of changing plaintext into ciphertext using a cryptographic algorithm and key

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.24
ephemeral key
a cryptographic key that is generated for each execution of a key establishment process and that meets other requirements of the key type (e.g., unique to each message or session). In some cases ephemeral keys are used more than once, within a single “session (e.g., broadcast applications) where the sender generates only one ephemeral key pair per message and the private key is combined separately with each recipient’s public key.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.25
global key
key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a DLMS®/COSEM Application Association, see also static symmetric key

3.2.26
hash function
a function that maps a bit string of arbitrary length to a fixed-length bit string. Approved hash functions satisfy the following properties:

1) One-way: It is computationally infeasible to find any input that maps to any pre-specified output, and
2) Collision resistant: It is computationally infeasible to find any two distinct inputs that map to the same output.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.27
hash value
the result of applying a hash function to information

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.28
initialization vector
IV
a vector used in defining the starting point of a cryptographic process

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.29
identification
the process of verifying the identity of a user, process, or device, usually as a prerequisite for granting access to resources in an IT system

[SOURCE: NIST SP 800-47:2002]

3.2.30
key
see cryptographic key

3.2.31
key agreement
a (pair-wise) key-establishment procedure in which the resultant secret keying material is a function of information contributed by both participants, so that neither party can predetermine the value of the secret keying material independently from the contributions of the other party. Contrast with key-transport.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.32
key-confirmation
a procedure to provide assurance to one party (the key-confirmation recipient) that another party (the key-confirmation provider) actually possesses the correct secret keying material and/or shared secret

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.33
key-derivation function
a function by which keying material is derived from a shared secret (or a key) and other information

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.34
key distribution
the transport of a key and other keying material from an entity that either owns the key or generates the key to another entity that is intended to use the key

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.35
key-encrypting key
a cryptographic key that is used for the encryption or decryption of other keys

Note 1 to entry: In DLMS®/COSEM it is the master key.

[SOURCE: NIST SP 800-57:2012, Part 1, modified by adding the Note]

3.2.36
key establishment
the procedure that results in keying material that is shared among different parties

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.37
key pair
a public key and its corresponding private key; a key pair is used with a public key algorithm

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.38
key revocation
a function in the lifecycle of keying material; a process whereby a notice is made available to affected entities that keying material should be removed from operational use prior to the end of the established cryptoperiod of that keying material

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.39
key-transport
a (pair-wise) key-establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver). Contrast with key agreement.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.40
key wrapping
a method of encrypting keying material (along with associated integrity information) that provides both confidentiality and integrity protection using a symmetric key

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.41
message authentication code
MAC
a cryptographic checksum on data that uses a symmetric key to detect both accidental and intentional modifications of data

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.42
message digest
the result of applying a hash function to a message. Also known as “hash value”.

[SOURCE: FIPS PUB 186-4:2013]

3.2.43
named curve
a set of ECDH domain parameters is also known as a "curve". A curve is a "named curve" if the domain parameters are well known and defined and can be identified by an Object Identifier; otherwise, it is called a "custom curve".

[SOURCE: RFC 5349]

3.2.44
nonce
a time-varying value that has at most an acceptably small chance of repeating. For example, the nonce may be a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.45
non-repudiation
a service that is used to provide assurance of the integrity and origin of data in such a way that the integrity and origin can be verified by a third party as having originated from a specific entity in possession of the private key of the claimed signatory

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.46
password
a string of characters (letters, numbers and other symbols) that are used to authenticate an identity or to verify access authorization or to derive cryptographic keys

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.47
plaintext
intelligible data that has meaning and can be understood without the application of decryption

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.48
private key
a cryptographic key, used with a public key cryptographic algorithm, which is uniquely associated with an entity and is not made public. In an asymmetric (public) cryptosystem, the private key is associated with a public key. Depending on the algorithm, the private key may be used, for example, to:

1) Compute the corresponding public key,
2) Compute a digital signature that may be verified by the corresponding public key,
3) Decrypt keys that were encrypted by the corresponding public key, or
4) Compute a shared secret during a key-agreement transaction.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.49
protected
ciphered and /or digitally signed. Protection may be applied to xDLMS APDUs and/or to COSEM data.

3.2.50
public key
a cryptographic key used with a public key cryptographic algorithm that is uniquely associated with an entity and that may be made public. In an asymmetric (public) cryptosystem, the public key is associated with a private key. The public key may be known by anyone and, depending on the algorithm, may be used, for example, to:

1) Verify a digital signature that is signed by the corresponding private key,
2) Encrypt keys that can be decrypted using the corresponding private key, or
3) Compute a shared secret during a key-agreement transaction.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.51
public-key certificate
a data structure that contains an entity’s identifier(s), the entity's public key (including an indication of the associated set of domain parameters) and possibly other information, along with a signature on that data set that is generated by a trusted party, i.e. a certificate authority, thereby binding the public key to the included identifier(s).

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.52
public key (asymmetric) cryptographic algorithm
a cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that determining the private key from the public key is computationally infeasible.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.53
Public Key Infrastructure
PKI
a framework that is established to issue, maintain and revoke public key certificates

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.54
receiver <key-transport>
the party that receives secret keying material via a key-transport transaction. Contrast with sender.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.55
revoke a certificate
to prematurely end the operational period of a certificate effective at a specific date and time

[SOURCE: NIST SP 800-32:2001]

3.2.56
Root Certification Authority
in a hierarchical Public Key Infrastructure, the Certification Authority whose public key serves as the most trusted datum (i.e., the beginning of trust paths) for a security domain

[SOURCE: NIST SP 800-32:2001]

3.2.57
secret key
a cryptographic key that is used with a secret key (symmetric) cryptographic algorithm that is uniquely associated with one or more entities and is not made public. The use of the term “secret” in this context does not imply a classification level, but rather implies the need to protect the key from disclosure

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.58
security services
mechanisms used to provide confidentiality, data integrity, authentication or non-repudiation of information

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.59
security strength
(also “bits of security”)
a number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.60
self-signed certificate
a public key certificate whose digital signature may be verified by the public key contained within the certificate. The signature on a self-signed certificate protects the integrity of the data, but does not guarantee authenticity of the information. The trust of self-signed certificates is based on the secure procedures used to distribute them.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.61
sender <key-transport>
the party that sends secret keying material to the receiver in a key-transport transaction. Contrast with receiver.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.62
session key
cryptographic key established for use for a relatively short period of time.

Note 1 to entry: In DLMS®/COSEM the dedicated key is a session key.

3.2.63
shared secret
a secret value that has been computed using a key agreement scheme and is used as input to a key-derivation function/method

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.64
signature generation
uses a digital signature algorithm and a private key to generate a digital signature on data

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.65
signature verification
uses a digital signature algorithm and a public key to verify a digital signature on data

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.66
signed data
data upon which a digital signature has been computed

3.2.67
static symmetric key
key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a DLMS®/COSEM Application Association

Note 1 to entry: In DLMS®/COSEM it is known as global key.

3.2.68
static key
a key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a cryptographic key establishment scheme. Contrast with an ephemeral key.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.69
Subordinate Certification Authority
in a hierarchical PKI, a Certification Authority (CA) whose certificate signature key is certified by another CA, and whose activities are constrained by that other CA

[SOURCE: NIST SP 800-32:2001]

3.2.70
symmetric key
a single cryptographic key that is used with a secret (symmetric) key algorithm

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.71
symmetric key algorithm
a cryptographic algorithm that uses the same secret key for an operation and its complement (e.g., encryption and decryption)

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.72
trust anchor
a public key and the name of a certification authority that is used to validate the first certificate in a sequence of certificates. The trust anchor public key is used to verify the signature on a certificate issued by a trust anchor certification authority. The security of the validation process depends upon the authenticity and integrity of the trust anchor. Trust anchors are often distributed as self-signed certificates.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.73
trusted party
a trusted party is a party that is trusted by an entity to faithfully perform certain services for that entity. An entity could be a trusted party for itself.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.74
trusted third party
a third party, such as a CA, that is trusted by its clients to perform certain services. (By contrast, in a key establishment transaction, the participants, parties U and V, are considered to be the first and second parties.)

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

3.2.75
X.509 certificate
the X.509 public-key certificate or the X.509 attribute certificate, as defined by the ISO/ITU-T X.509 standard. Most commonly (including in this document), an X.509 certificate refers to the X.509 public-key certificate.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.2.76
X.509 public key certificate
a digital certificate containing a public key for entity and a name for the entity, together with some other information that is rendered unforgeable by the digital signature of the certification authority that issued the certificate, encoded in the format defined in the ISO/ITU-T X.509 standard.

[SOURCE: NIST SP 800-57:2012, Part 1]

3.3 Definitions and abbreviations related to the Galois/Counter Mode

The source of the definitions 3.3.1 to 3.3.13 abbreviations and symbols in this subclause is NIST SP 800-38D:2007.

3.3.1
Additional Authenticated Data
AAD
input data to the authenticated encryption function that is authenticated but not encrypted

3.3.2
authenticated decryption
function of GCM in which the ciphertext is decrypted into the plaintext, and the authenticity of the ciphertext and the AAD are verified

3.3.3
authenticated encryption
function of GCM in which the plaintext is encrypted into the ciphertext and an authentication tag is generated on the AAD and the ciphertext

3.3.4
authentication tag
Tag, T
cryptographic checksum on data that is designed to reveal both accidental errors and the intentional modification of the data

3.3.5
block cipher
parameterized family of permutations on bit strings of a fixed length; the parameter that determines the permutation is a bit string called the key

3.3.6
ciphertext
encrypted form of the plaintext

3.3.7
fixed field
in the deterministic construction of IVs, the field that identifies the device or context for the instance of the authenticated encryption function

3.3.8
fresh
for a newly generated key, the property of being unequal to any previously used key

3.3.9
GCM
Galois/Counter Mode

3.3.10
initialization Vector
IV
nonce that is associated with an invocation of authenticated encryption on a particular plaintext and AAD

Note 1 to entry: For the purposes of this standard, the invocation field is the invocation counter.

3.3.11
invocation field
in the deterministic construction of IVs, the field that identifies the sets of inputs to the authenticated encryption function in a particular device or context

3.3.12
key
parameter of the block cipher that determines the selection of the forward cipher function from the family of permutations

3.3.13
plaintext
P
input data to the authenticated encryption function that is both authenticated and encrypted

3.3.14
security control byte
SC
byte that provides information on the ciphering applied

3.3.15
security header
SH
concatenation of the security control byte SC and the invocation counter: $SH = SC \parallel IC$.

1205 ## 3.4 General abbreviations

Abbreviation	Meaning
.cnf	.confirm service primitive
.ind	.indication service primitive
.req	.request service primitive
.res	.response service primitive
AA	Application Association
AARE	A-Associate Response – an APDU of the ACSE
AARQ	A-Associate Request – an APDU of the ACSE
ACPM	Association Control Protocol Machine
ACSE	Association Control Service Element
AE	Application Entity
AES	Advanced Encryption Standard
AL	Application Layer
AP	Application Process
APDU	Application Layer Protocol Data Unit
API	Application Programming Interface
ASE	Application Service Element
ASO	Application Service Object
ATM	Asynchronous Transfer Mode
A-XDR	Adapted Extended Data Representation
base_name	The short_name corresponding to the first attribute (“logical_name”) of a COSEM object
BER	Basic Encoding Rules
BD	Block Data
BN	Block Number
BNA	Block Number Acknowledged
BS	Bit string
BTS	Block Transfer Streaming
BTW	Block Transfer Window
CA	Certification Authority
CF	Control Function
CL	Connectionless
class_id	COSEM interface class identification code
CMP	Certificate Management Protocol. Refer to RFC 4210.
CO	Connection-oriented
COSEM	Companion Specification for Energy Metering
COSEM_on_IP	The TCP-UDP/IP based COSEM communication profile
CRC	Cyclic Redundancy Check
CRL	Certificate revocation list. Refer to RFC 5280.
CSR	Certificate Signing Request
DCE	Data Communication Equipment (communications interface or modem)
DCS	Data Collection System
DISC	Disconnect (a HDLC frame type)
DLMS®	Device Language Message Specification
DM	Disconnected Mode (a HDLC frame type)
DSA	Digital Signature Algorithm specified in FIPS PUB 186-4:2013

Abbreviation	Meaning
DSAP	Data Link Service Access Point
DSO	Energy Distribution System Operator
DTE	Data Terminal Equipment (computers, terminals or printers)
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman key agreement protocol
ECDSA	Elliptic Curve Digital Signature Algorithm specified in ANSI X9.62 and FIPS PUB 186-4:2013
ECP	Elliptic Curve Point
EUI-64	64-bit Extended Unique Identifier
FCS	Frame Check Sequence
FDDI	Fibre Distributed Data Interface
FE	Field Element (in relation with public key algorithms)
FIPS	Federal Information Processing Standard
FRMR	Frame Reject (a HDLC frame type)
FTP	File Transfer Protocol
GAK	Global Authentication Key
GBEK	Global Broadcast Encryption Key
GBT	General Block Transfer
GCM	Galois/Counter Mode (GCM), an algorithm for authenticated encryption with associated data
GMAC	A specialization of GCM for generating a message authentication code (MAC) on data that is not encrypted
GMT	Greenwich Mean Time
GSM	Global System for Mobile communications
GUEK	Global Unicast Encryption Key
GW	Gateway
HCS	Header Check Sequence
HDLC	High-level Data Link Control
HES	Head End System, also known as Data Collection System NOTE The HES may be owned by the energy provider or the utility
HHU	Hand Held Unit
HLS	High Level Security (COSEM)
HMAC	Keyed-Hash Message Authentication Code specified in FIPS 198-1
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
I	Information (a HDLC frame type)
IANA	Internet Assigned Numbers Authority
IC	Interface Class
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
IV	Initialization Vector
KEK	Key Encrypting Key
LAN	Local Area Network
LB	Last Block

Abbreviation	Meaning
LDN	Logical Device Name
LLC	Logical Link Control (Sublayer)
LLS	Low Level Security
LNAP	Local Network Access Point
LPDU	LLC Protocol Data Unit
L-SAP	LLC sublayer Service Access Point
LSB	Least Significant Bit
LSDU	LLC Service Data Unit
m	mandatory, used in conjunction with attribute and method definitions
MAC	Medium Access Control (sublayer)
MAC	Message Authentication Code (cryptography)
MIB	Management Information Base
MSAP	MAC sublayer Service Access Point (in the HDLC based profile, it is equal to the HDLC address)
MSB	Most Significant Bit
MSC	Message Sequence Chart
MSDU	MAC Service Data Unit
N(R)	Receive sequence Number
N(S)	Send sequence Number
NDM	Normal Disconnected Mode
NIST	National Institute of Standards and Technology
NNAP	Neighbourhood Network Access Point
NRM	Normal Response Mode
o	optional, used in conjunction with attribute and method definitions
OBIS	Object Identification System
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OOB	Out of Band
OS	Octet string
OSI	Open System Interconnection
OTA	Over The Air
P/F	Poll/Final
PAR	Positive Acknowledgement with Retransmission
PDU	Protocol data unit
PhL	Physical Layer
PHSDU	PH SDU
PKCS	Public Key Cryptography Standard, established by RSA Laboratories
PKI	Public Key Infrastructure
PLC	Power line carrier
PPP	Point-to-Point Protocol
PSDU	Physical layer Service Data Unit
PSTN	Public Switched Telephone Network
RA	Registration Authority
RLRE	A-Release Response – an APDU of the ACSE

Abbreviation	**Meaning**
RLRQ	A-Release Request – an APDU of the ACSE
RNG	Random Number Generator
RNR	Receive Not Ready (a HDLC frame type)
RR	Receive Ready (a HDLC frame type)
RSA	Algorithm developed by Rivest, Shamir and Adelman; specified in ANS X9.31 and PKCS #1.
SAP	Service Access Point
SDU	Service Data Unit
SHA	Secure Hash Algorithm; specified in FIPS PUB 180-4:2012.
SNMP	Simple Network Management Protocol
SNRM	Set Normal Response Mode (a HDLC frame type)
STR	Streaming
tbsCertificate	To be signed certificate
TCP	Transmission Control Protocol
TDEA	Triple Data Encryption Algorithm
TL	Transport Layer
TPDU	Transport Layer Protocol Data Unit
TWA	Two Way Alternate
UA	Unnumbered Acknowledge (a HDLC frame type)
UDP	User Datagram Protocol
UI	Unnumbered Information (a HDLC frame type)
UNC	Unbalanced operation Normal response mode Class
USS	Unnumbered Send Status
V(R)	Receive state Variable
V(S)	Send state Variable
VAA	Virtual Application Association
WPDU	Wrapper Protocol Data Unit
xDLMS ASE	Extended DLMS® Application Service Element
See also list of abbreviations specific to a cryptographic algorithm in the relevant clauses.	

1207 3.5 Symbols related to the Galois/Counter Mode

Symbol	Meaning
A	Additional Authenticated Data, AAD
AK	Authentication key, a parameter that is part of the AAD
C	Ciphertext
EK	Encryption key, i.e. the block cipher key
IC	Invocation counter, part of the initialization vector. See also invocation field.
IV	Initialization Vector
$\mathrm{len}(X)$	The bit length of the bit string X.
$\mathrm{LEN}(X)$	The octet length of the octet string X.
P	Plaintext
SC	Security Control Byte
SH	Security Header
$Sys\text{-}T$	System title
T	Authentication tag
t	The bit length of the authentication tag. NOTE This is the same as $\mathrm{len}(T)$
$X \parallel Y$	Concatenation of two strings, X and Y.

1208 3.6 Symbols related the ECDSA algorithm

Symbol	Meaning
d	The ECDSA private key, which is an integer in the interval $[1, n - 1]$.
$Q = (x_Q, y_Q)$	An ECDSA public key. The coordinates x_Q and y_Q are integers in the interval $[0, q - 1]$, and $Q = dG$.
k	The ECDSA per-message secret number, which is an integer in the interval $[1, n - 1]$.
r	One component of an ECDSA digital signature. It is an integer in $[1, n - 1]$. See the definition of (r, s).
s	One component of an ECDSA digital signature. It is an integer in $[1, n - 1]$. See the definition of (r, s).
(r, s)	An ECDSA digital signature, where r and s are the digital signature components.
M	The message that is signed using the digital signature algorithm.
$\mathrm{Hash}(M)$	The result of a hash computation (message digest or hash value) on message M using an approved hash function.

1209

3.7 Symbols related to the key agreement algorithms

Symbol	Meaning
$d_{e,U}$, $d_{e,V}$	Party U's and Party V's ephemeral private keys. These are integers in the range [1, n-1].
$d_{s,U}$, $d_{s,V}$	Party U's and Party V's static private keys. These are integers in the range [1, n-1].
$\mathrm{ID_U}$	The identifier of Party U (the initiator)
$\mathrm{ID_V}$	The identifier of Party V (the responder)
$Q_{e,U}$, $Q_{e,V}$	Party U's and Party V's ephemeral public keys. These are points on the elliptic curve defined by the domain parameters.
$Q_{s,U}$, $Q_{s,V}$	Party U's and Party V's static public keys. These are points on the elliptic curve defined by the domain parameters.
U, V	Represent the two parties in a (pair-wise) key establishment scheme.
Z	A shared secret (represented as a byte string) that is used to derive secret keying material using a key derivation method. *Source: NIST SP 800-56A Rev. 2: 2013*

4 Overview of DLMS®/COSEM

4.1 Information exchange in DLMS®/COSEM

4.1.1 General

This subclause 4.1 introduces the main concepts of information exchange in DLMS®/COSEM.

The objective of DLMS®/COSEM is to specify a standard for a business domain oriented interface object model for metering devices and systems, as well as services to access the objects. Communication profiles to transport the messages through various communication media are also specified.

The term "metering devices" is an abstraction; consequently "metering device" may be any type of device for which this abstraction is suitable.

The COSEM object model is specified in IEC 62056-6-2:2021. The COSEM objects provide a view of the functionality of metering devices through their communication interfaces.

This International Standard specifies the DLMS®/COSEM application layer and the rules for specifying DLMS®/COSEM communication profiles; see Annex A.

The key characteristics of data exchange using DLMS®/COSEM are the following:

- metering devices can be accessed by various parties: clients and third parties;
- mechanisms to control access to the resources of the metering device are provided; these mechanisms are made available by the DLMS®/COSEM AL and the COSEM objects ("Association SN / LN" object, "Security setup" object);
- security and privacy is ensured by applying cryptographical protection to xDLMS messages and to COSEM data;
- low overhead and efficiency is ensured by various mechanisms including selective access, compact encoding and compression;
- at a metering site, there may be single or multiple metering devices. In the case of multiple metering devices at a metering site, a single access point can be made available;
- data exchange may take place either remotely or locally. Depending on the capabilities of the metering device, local and remote data exchange may be performed simultaneously without interfering with each other;

- various communication media can be used on local networks (LN), neighbourhood networks (NN) and wide area networks (WAN).

The key element to ensure that the above requirements are met is the Application Association (AA) – determining the contexts of the data exchange – provided by the DLMS®/COSEM AL. For details, see the relevant clauses below.

4.1.2 Communication model

DLMS®/COSEM uses the concepts of the Open Systems Interconnection (OSI) model to model information exchange between meters and data collection systems.

NOTE Information in this context comprises xDLMS messages and COSEM data.

Concepts, names and terminology used below relate to the OSI reference model described in ISO/IEC 7498-1:1994. Their use is outlined in this clause and further developed in other clauses.

Application functions of metering devices and data collection systems are modelled by application processes (APs).

Communication between APs is modelled by communication between application entities (AEs). An AE represents the communication functions of an AP. There may be multiple sets of OSI communication functions in an AP, so a single AP may be represented by multiple AEs. However, each AE represents a single AP. An AE contains a set of communication capabilities called application service elements (ASEs). An ASE is a coherent set of integrated functions. These ASEs may be used independently or in combination. See also 4.2.2.

Data exchange between data collection systems and metering devices is based on the client/server model where data collection systems play the role of the client and metering devices play the role of the server. The client sends service requests to the server which sends service responses. In addition the server may initiate unsolicited service requests to inform the client about events or to send data on pre-configured conditions. See also 4.1.6.

In general, the client and the server APs are located in separate devices. Therefore, message exchange takes place via a protocol stack as shown in Figure 1.

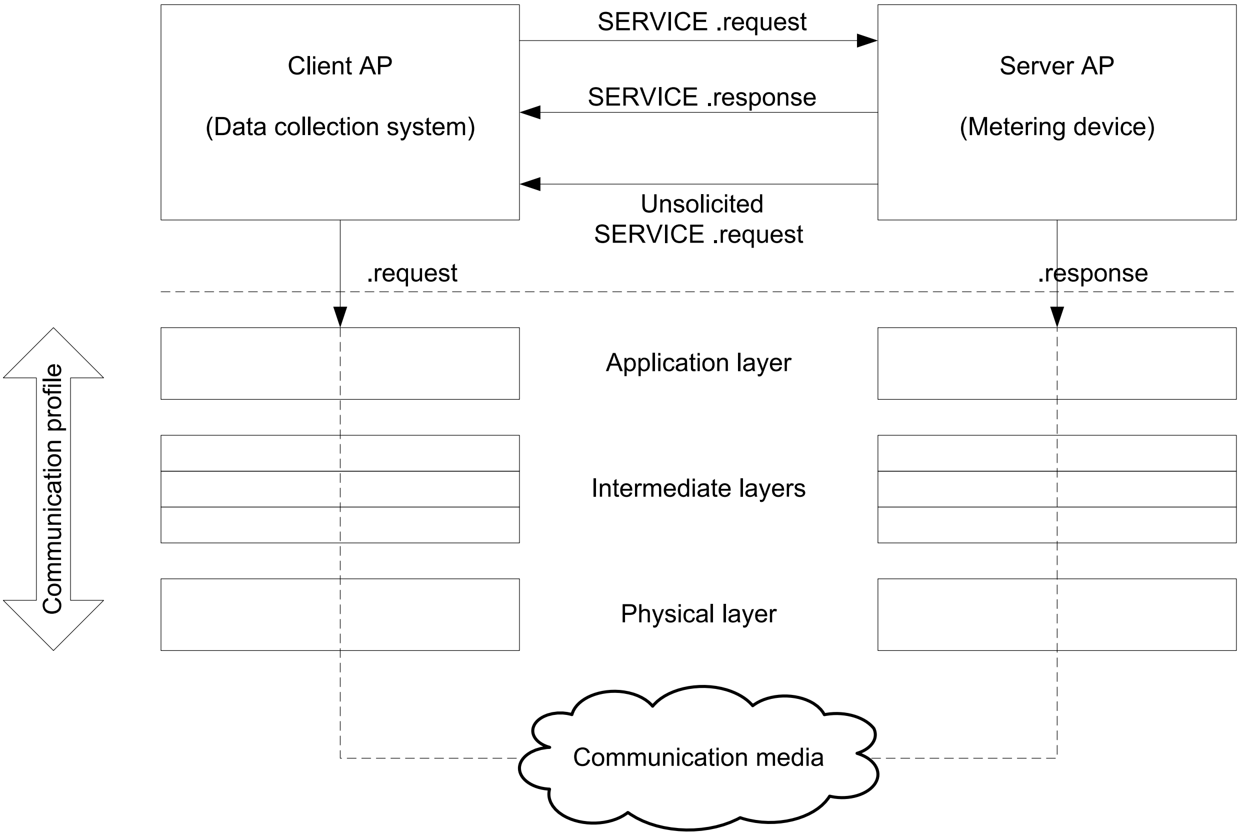


Figure 1 – Client–server model and communication protocols

4.1.3 Naming and addressing

4.1.3.1 General

Naming and addressing are important aspects in communication systems. A name identifies a communicating entity. An address identifies where that entity can be found. Names are mapped to addresses; this is known as the process of binding. Figure 2 shows the main elements of naming and addressing in DLMS®/COSEM.

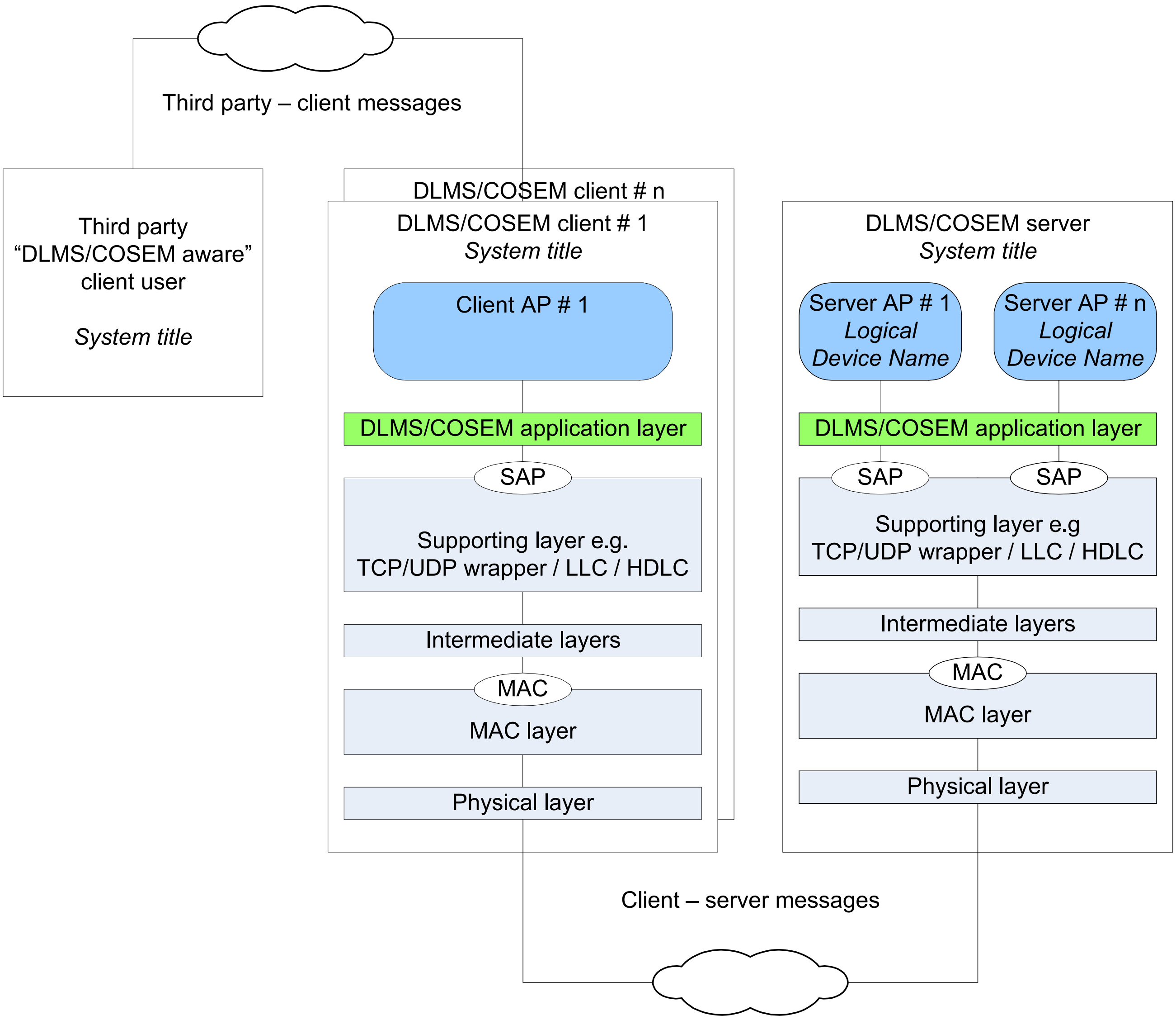


Figure 2 – Naming and addressing in DLMS®/COSEM

4.1.3.2 Naming

DLMS®/COSEM entities, including clients, servers as well as third party systems shall be uniquely named by their *system title*. System titles shall be permanently assigned.

Server physical devices may host one or more logical devices (LDs). LDs shall be uniquely identified by their Logical Device Name (LDN). LDs hosted by the same physical device share the system title. System titles are specified in 4.1.3.4. Logical device names are specified in 4.1.3.5.

4.1.3.3 Addressing

Each physical device shall have an appropriate address. It depends on the communication profile and may be a phone number, a MAC address, an IP network address or a combination of these.

NOTE For example, in the case of the 3-layer, connection-oriented, HDLC based communication profile, the lower HDLC address is the MAC address.

Physical device addresses may be pre-configured or may be assigned during a registration process, which also involves binding between the addresses and the system titles.

Each DLMS®/COSEM client and each server – a COSEM logical device – is bound to a Service Access Point (SAP). The SAPs reside in the supporting layer of the DLMS®/COSEM AL. Depending on the communication profile the SAP may be a TCP-UDP/IP wrapper address, an upper HDLC address, an LLC address etc. On the server side, this binding is modelled by the “SAP Assignment” IC; see IEC 62056-6-2:2021, 4.4.5.

The values of the SAPs on the client and the server side are specified in Table 1. The length of the SAPs depends on the communication profile.

Table 1 – Client and server SAPs

Client SAPs	
No-station	0x00
Client Management Process / CIASE [1]	0x01
Public Client	0x10
Open for client AP assignment	0x02 …0x0F
	0x11 and up
Server SAPs	
No-station / CIASE [1]	0x00
Management Logical Device	0x01
Reserved for future use	0x02…0x0F
Open for server SAP assignment	0x10 and up
All-station (Broadcast)	Communication profile specific
[1] In the case of the DLMS®/COSEM S-FSK PLC profile, see IEC 62056-8-3.	
NOTE Depending on the supporting layer, the SAPs may be represented on one or more bytes.	

4.1.3.4 System title

The system title *Sys-T* shall uniquely identify each DLMS®/COSEM entity that may be server, a client or a third party that can access servers via clients. The system title:

- shall be 8 octets long;
- shall be unique.

The leading (i.e., the 3 leftmost) octets should hold the three-letter manufacturer ID[2]. This is the same as the leading three octets of the Logical Device Name, see 4.1.3.5. The remaining 5 octets shall ensure uniqueness.

NOTE It can be derived for example from the last 12 digits of the manufacturing number, up to 999 999 999 999. This value converts to 0xE8D4A50FFF. Values above this, up to 0xFFFFFFFFFF (decimal 1 099 511 627 775) can also be used, but these values cannot be mapped to the last 12 digits of the manufacturing number.

Project specific companion specifications may specify a different structure. In that case, the details should be specified by the naming authority designated as such for the project.

[2] Administered by the FLAG Association in co-operation with the DLMS UA.

The use of the system title in cryptographic protection of xDLMS messages and COSEM data is further specified in 5.3 and 5.7.

Before the cryptographic security algorithms can be used – this requires a ciphered application context – the peers have to exchange system titles. The following possibilities are available:

- during the communication media specific registration process. For example, when the S-FSK PLC profile is used, system titles are exchanged during the registration process using the CIASE protocol; see IEC 62056-8-3;
- in all communication profiles, system titles may be exchanged during AA establishment using the COSEM-OPEN service, see 6.2, carried the AARQ / AARE APDU. If the system titles sent / received during AA establishment are not the same as the ones exchanged during the registration process, the AA shall be rejected;
- by writing the *client_system_title* attribute and by reading the *server_system_title* attribute of "Security setup" objects, see IIEC 62056-6-2:2021, 4.4.7.

In the case of broadcast communication, only the client sends the system title to the server.

4.1.3.5 Logical Device Name

Logical Device Name (LDN) shall be as specified in IEC 62056-6-2:2021, 4.1.4.6.

4.1.3.6 Client user identification

The client user identification mechanism allows a server to distinguish between different users on the client side and to log their activities accessing the meter. It is specified in IEC 62056-6-2:2021, 4.4.2. Naming of client users is outside the scope of this International Standard.

4.1.4 Connection oriented operation

The DLMS®/COSEM AL is connection oriented. See also 4.2.3.

A communication session consists of three phases, as it is shown in Figure 3:

- first, an application level connection, called Application Association (AA), is established between a client and a server AE; see also 4.2.3. Before initiating the establishment of an AA, the peer PhLs of the client and server side protocol stacks have to be connected. The intermediate layers may have to be connected or not. Each layer, which needs to be connected, may support one or more connections simultaneously;
- once the AA is established, message exchange can take place;
- at the end of the data exchange, the AA is released.

Figure 3 – A complete communication session in the CO environment

For the purposes of very simple devices, one-way communicating devices, and for multicasting and broadcasting pre-established AAs are also allowed. For such AAs the full communication session may include only the message exchange phase: it can be considered that the connection establishment phase has been already done somewhere in the past. Pre-established AAs cannot be released. See also 7.2.4.4.

4.1.5 Application associations

4.1.5.1 General

Application Associations (AAs) are logical connections between a client and a server AE. AAs may be established on the request of a client using the services of the connection-oriented ACSE of the AL or may be pre-established. They may be confirmed or unconfirmed. See also 4.2.3.

NOTE 1 A pre-established AA can be considered to have been established in the past.

NOTE 2 Servers cannot initiate the establishment of an AA.

A COSEM logical device may support one or more AAs, each with a different client. Each AA determines the contexts in which information exchange takes place.

A confirmed AA is proposed by the client and accepted by the server provided that:

- the user of the client is known by the server, see 4.1.3.6;
- the application context proposed by the client – see 4.1.5.2 – is acceptable for the server;
- the authentication mechanism proposed by the client – see 4.1.5.3 – is acceptable for the server and the authentication is successful;
- the elements of the xDLMS context – see 4.1.5.4 – can be successfully negotiated between the client and the server.

An unconfirmed AA is also proposed by a client with the assumption that the server will accept it. No negotiation takes place. Unconfirmed AAs are useful for sending broadcast messages from the client to servers.

AAs are modelled by COSEM “Association SN / LN” objects that hold the SAPs identifying the associated partners, the name of the *application context*, the name of the *authentication mechanism*, and the *xDLMS context.*

The "Association SN / LN" objects also determine a specific set of access rights to COSEM object attributes and methods and they point to (reference) a "Security setup" object that hold the elements of the security context. The access rights and the security context may be different in each AA. These objects are specified in IEC 62056-6-2:2021, 4.4.3, 4.4.4.

4.1.5.2 Application context

The application context determines:

- the set of Application Service Elements (ASEs) present in the AL;
- the referencing style of COSEM object attributes and methods: short name (SN) referencing or logical name (LN) referencing. See also 4.2.4.3.1;
- the transfer syntax;
- whether ciphering is used or not.

Application contexts are identified by names, see 7.2.2.2.

4.1.5.3 Authentication

In communication systems entity authentication is a fundamentally important security service. The goal of entity authentication is to establish whether the claimant of a certain identity is in fact who it claims to be. In order to achieve this goal, there should be a pre-existing relation which links the entity to a secret.

In DLMS®/COSEM, authentication takes place during AA establishment.

In confirmed AAs either the client (unilateral authentication) or both the client and the server (mutual authentication) can authenticate itself.

In an unconfirmed AA, only the client can authenticate itself.

In pre-established AAs, authentication of the communicating partners is not available.

Once the AA is established, COSEM object attributes and methods can be accessed using xDLMS services subject to the prevailing security context and access rights in the given AA.

The authentication mechanisms are specified in 5.2.2.2. The authentication mechanisms are identified by names, see 7.2.2.3.

4.1.5.4 xDLMS context

The xDLMS context determines the set of xDLMS services and capabilities that can be used in a given AA. See 4.2.4.

4.1.5.5 Security context

The security context is relevant when the application context stipulates ciphering. It comprises the security suite, the security policy, the security keys and other security material. See also 5.2.3. It is managed by "Security setup" objects.

4.1.5.6 Access rights

Access rights determine the rights of the client(s) to access COSEM object attributes and methods within an AA. The set of access rights depend on the role of the client and is pre-configured in the server. See also 5.2.4.

NOTE The roles and the related access rights are subject to project specific companion specifications. Examples for roles are meter reader, meter service / communication service / energy provider, manufacturer, end user, etc.

4.1.6 Messaging patterns

The messaging patterns available between a DLMS®/COSEM client and server are shown in Figure 4.

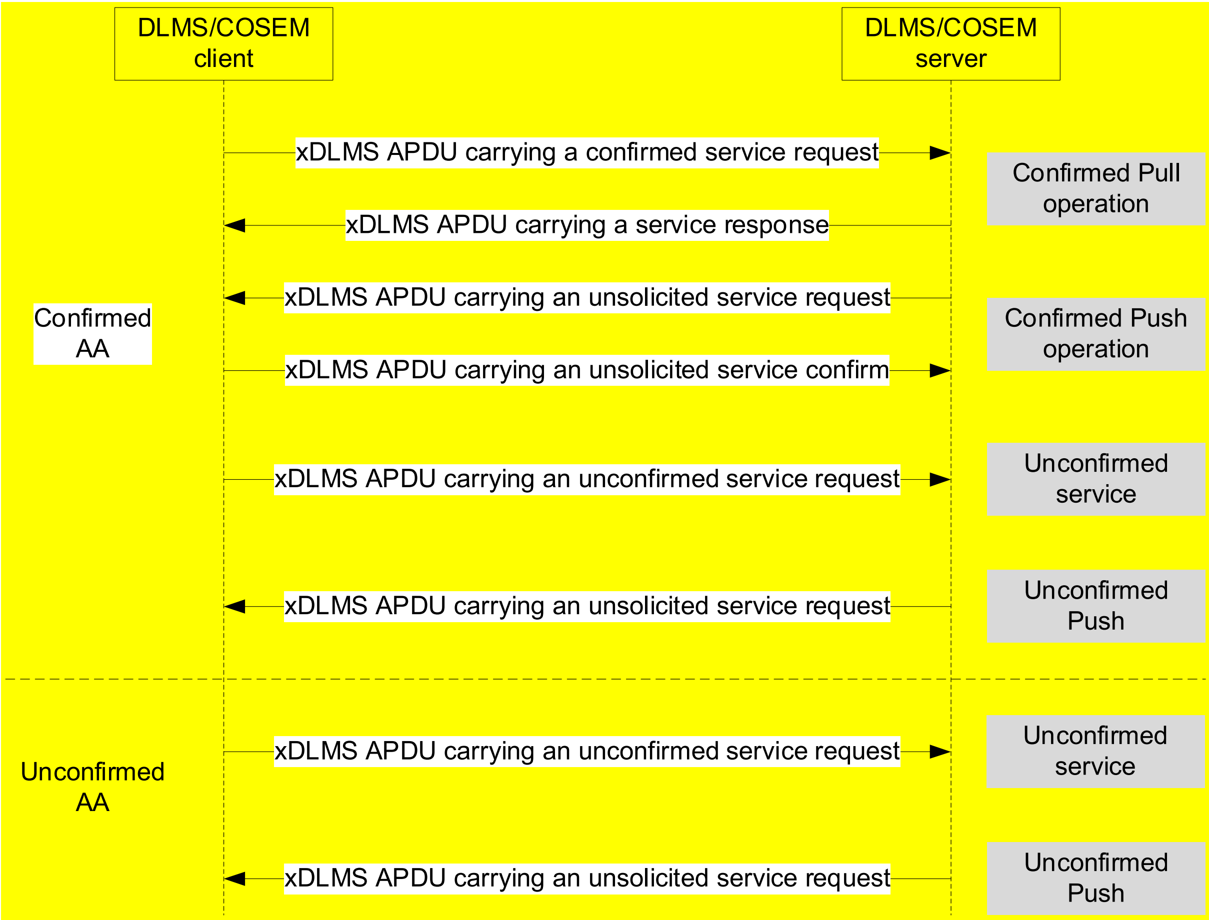


Figure 4 – DLMS®/COSEM messaging patterns

In confirmed AAs:

- the client can send confirmed service requests and the server responds: *pull operation*;
- the client can send unconfirmed service requests. The server does not respond;
- the server can send unsolicited service requests to the client: *push operation*.

NOTE The unsolicited services may be InformationReport (with SN referencing), EventNotification (with LN referencing) or DataNotification (used both with SN and LN referencing).

In unconfirmed AAs:

- only the client can initiate service requests and only unconfirmed ones. The server cannot respond and it cannot initiate service requests.

4.1.7 Data exchange between third parties and DLMS®/COSEM servers

Third parties – that are outside the DLMS®/COSEM client-server relationship – may also exchange information with servers, using a client as a broker. To support end-to-end security, such third parties shall be “DLMS®/COSEM aware” meaning that they shall be able to send messages to the client that contain properly formatted xDLMS APDUs carrying properly formatted COSEM data, and that they shall be able to process messages received from the server via the client. See also 5.2.5, Figure 14.

Messages from the server to the third party may be solicited or unsolicited.

4.1.8 Communication profiles

Communication profiles specify how the DLMS®/COSEM AL and the COSEM data model modelling the Application Process (AP) are supported by the lower, communication media specific protocol layers.

Communication profiles comprise a number of protocol layers. Each layer has a distinct task and provides services to its upper layer and uses services of its supporting layer(s). The client and server COSEM APs use the services of the highest protocol layer, that of the DLMS®/COSEM AL. This is the only protocol layer containing COSEM specific element(s): the xDLMS ASE; see 4.2.4. It may be supported by any layer capable of providing the services required by the DLMS®/COSEM AL. The number and type of lower layers depend on the communication media used.

A given set of protocol layers with the DLMS®/COSEM AL and the COSEM object model on top constitutes a particular DLMS®/COSEM communication profile. Each profile is characterized by the protocol layers included and their parameters.

Figure 5 shows a generic DLMS®/COSEM communication profile, including:

- the COSEM object model modelling the Application Process. For each communication media, media-specific setup interface classes are specified;
- the DLMS®/COSEM application layer;
- the DLMS®/COSEM transport layer, present in internet capable profiles;
- the convergence layers that bind the MAC layer to the DLMS®/COSEM AL either directly or through the DLMS®/COSEM transport layer;
- the media specific physical and MAC layers; and
- the connection managers.

A single physical device may support more than one communication profile to allow data exchange using various communication media. In such cases it is the task of the client side AP to decide which communication profile should be used.

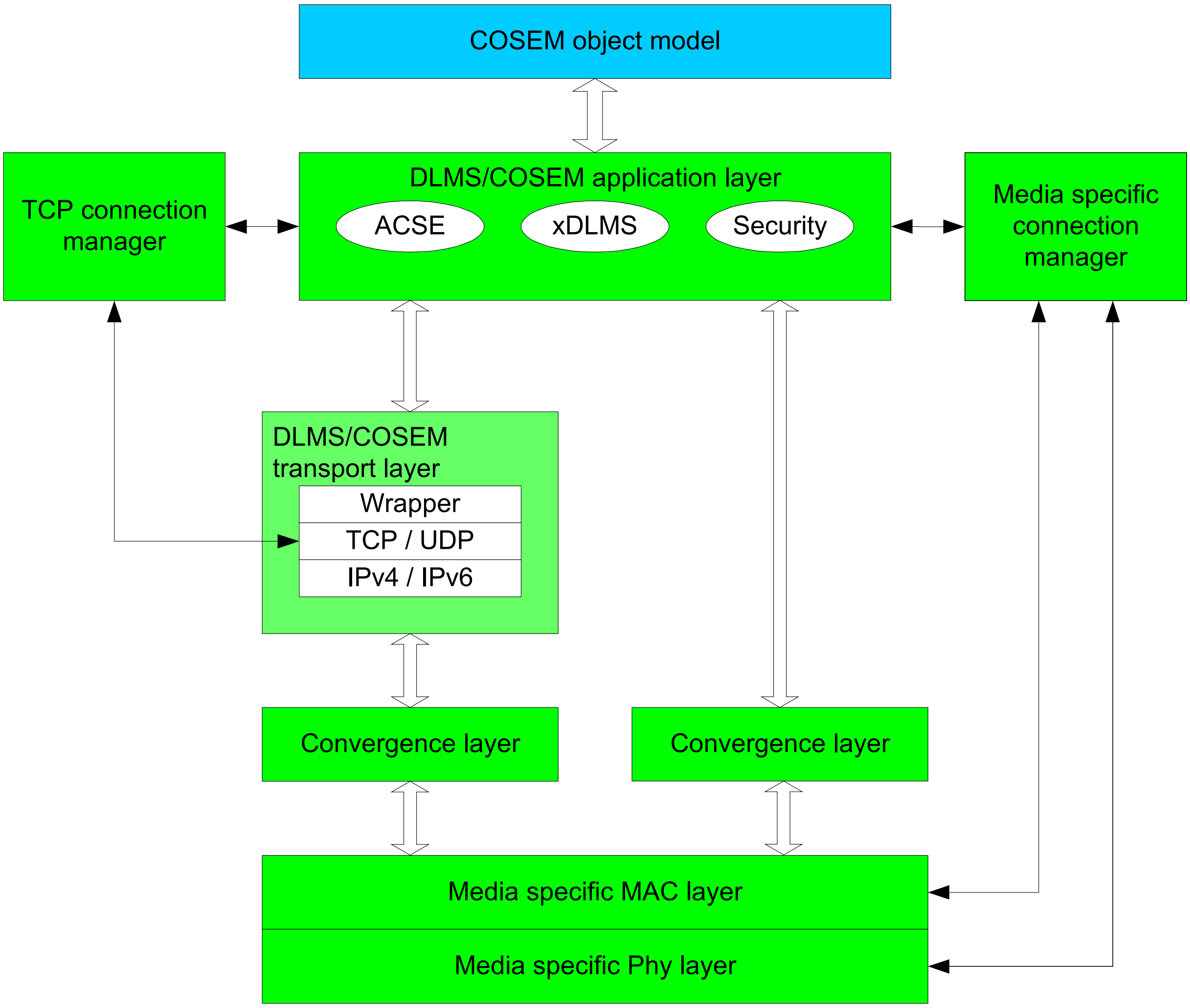


Figure 5 – DLMS®/COSEM generic communication profile

Using the DLMS®/COSEM application layer in various communications profiles Communication is specified in Annex A. Communication profile standards are specified in other parts of the IEC 62056 DLMS®/COSEM suite:

- the 3-layer, connection-oriented, HDLC based communication profile, is specified in IEC 62056-7-6;
- the TCP-UDP/IP based communication profiles (COSEM_on_IP), is specified in IEC 62056-9-7;
- the S-FSK PLC profile, is specified in IEC 62056-8-3.
- the wired and wireless M-Bus profile is specified in IEC 62056-7-3:—.
- the LPWAN profile that can be used over LoRaWAN networks, specified in IEC xxxxx;
- the Wi-SUN profile specified in IEC xxxxx

NOTE Further communication profiles may be specified in the future.

4.1.9 Model of a DLMS®/COSEM metering system

Figure 6 shows a model of a DLMS®/COSEM metering system.

Metering equipment are modelled as a set of logical devices, hosted in a single physical device. Each logical device represents a server AP and models a subset of the functionality of the metering equipment as these are seen through its communication interfaces. The various functions are modelled using COSEM objects.

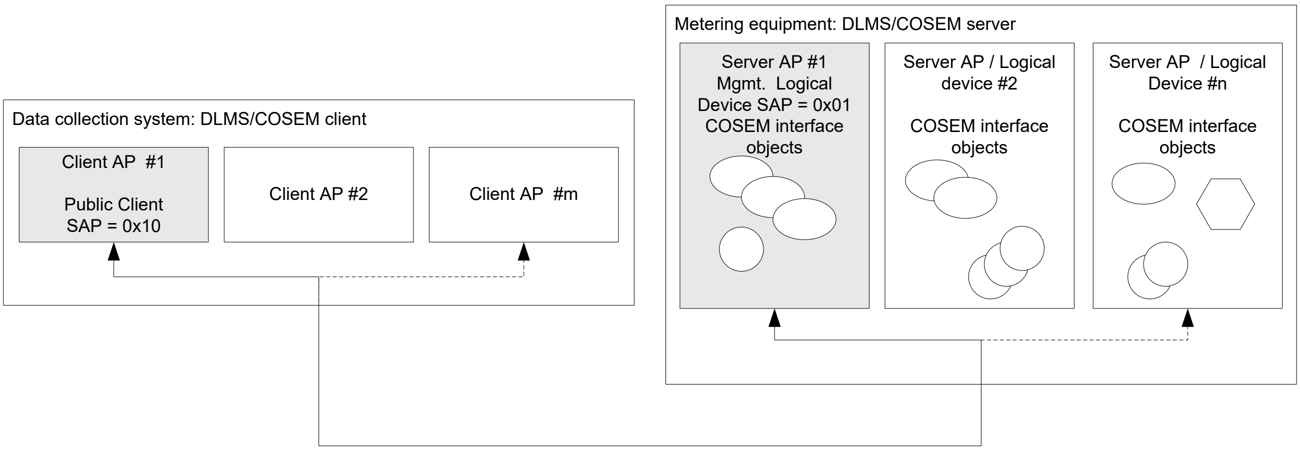


Figure 6 – Model of a DLMS®/COSEM metering system

Data collection systems are modelled as a set of client APs that may be hosted by one or several physical devices. Each client AP may have different roles and access rights, granted by the metering equipment.

The Public Client and the Management Logical Device APs have a special role and they shall be always present.

See more in IEC 62056-6-2:2021, 4.1.7 and 4.1.8.

4.1.10 Model of DLMS®/COSEM servers

Figure 7 shows the model of two DLMS®/COSEM servers as an example. One of them uses a 3-layer, CO, HDLC based communication profile, and the other one uses a TCP-UDP/IP based communication profile.

The metering equipment on the left hand side comprises "n" logical devices and supports the 3-layer, CO, HDLC based communication profile.

The DLMS®/COSEM AL is supported by the HDLC based data link layer. Its main role is to provide a reliable data transfer between the peer layers. It also provides addressing of the logical devices in such a way, that each logical device is bound to a single HDLC address. The Management Logical Device is always bound to the address 0x01. To allow creating a local network so that several metering devices at a given metering site can be reached through a single access point, another address, the physical address is also provided by the data link layer. The logical device addresses are referred to as upper HDLC addresses, while the physical device address is referred to as a lower HDLC address. See also IEC 62056-7-6.

The PhL supporting the data link layer provides serial bit transmission between physical devices hosting the client and server applications. This allows using various interfaces, like RS 232, RS 485, 20 mA current loop, etc. to transfer data locally through PSTN and GSM networks etc.

The metering equipment on the right hand side comprises "m" logical devices.

The DLMS®/COSEM AL is supported by the DLMS®/COSEM TL, comprising the internet TCP or UDP layer and a wrapper. The main role of the wrapper is to adapt the OSI-style service set, provided by the DLMS®/COSEM TL to and from TCP and UDP function calls. It also provides addressing for the logical devices, binding them to a SAP called wrapper port. The Management Logical Device is always bound to wrapper port 0x01. Finally, the wrapper provides information about the length of the APDUs transmitted, to help the peer to recognise the end of the APDU. This is necessary due the streaming nature of TCP.

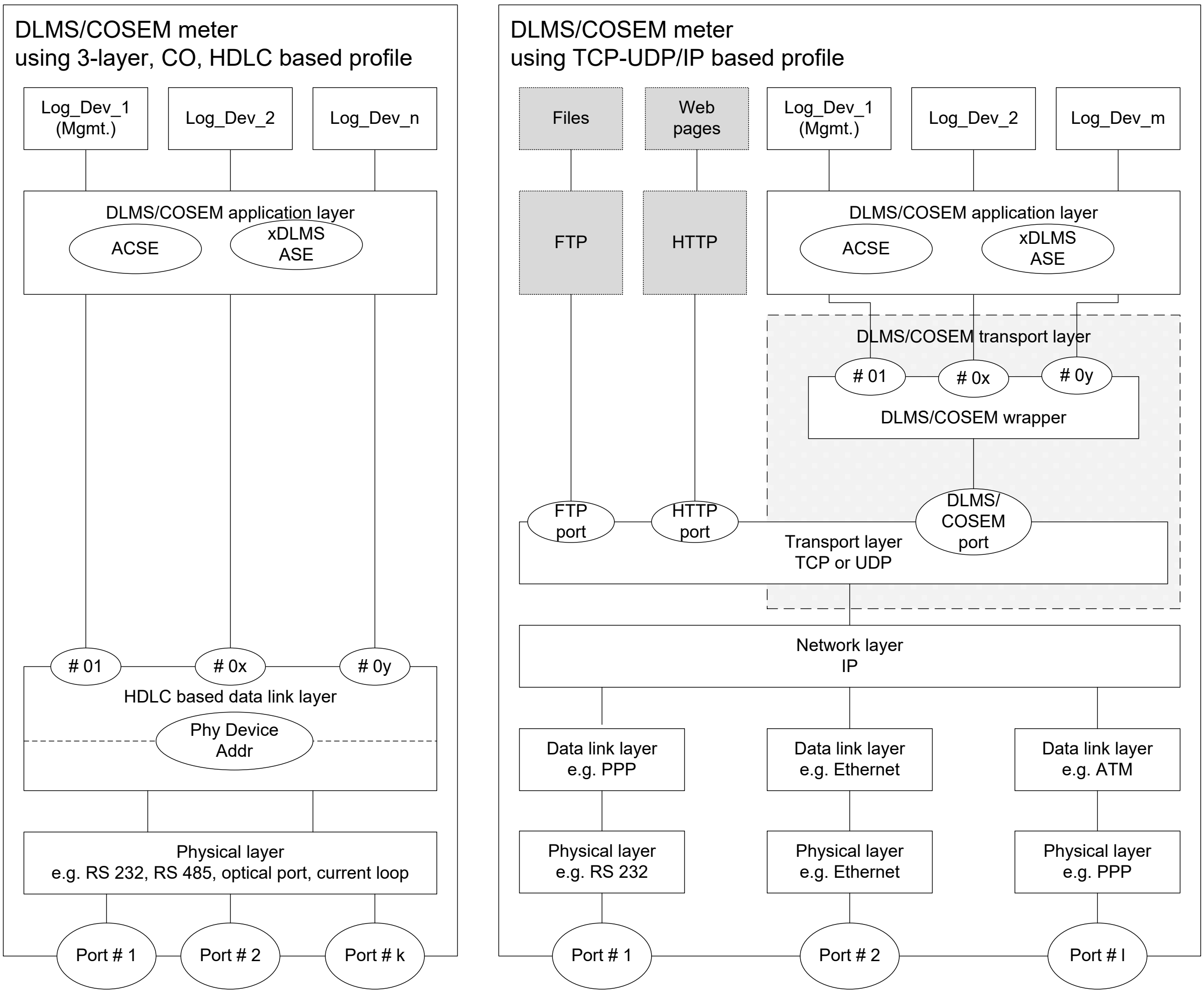


Figure 7 – DLMS®/COSEM server model

Through the wrapper, the DLMS®/COSEM AL is bound to a TCP or UDP port number, which is used for the DLMS®/COSEM application. The presence of the TCP and UDP layers allows incorporating other internet applications, like FTP or HTTP, bound to their standard ports respectively.

The TCP layer is supported by the IP layer, which is in turn may be supported by any set of lower layers depending on the communication media to be used (for example Ethernet, PPP, IEEE 802, or IP-capable PLC lower layers, etc.).

Obviously, in a single server it is possible to implement several protocol stacks, with the common DLMS®/COSEM AL being supported by distinct sets of lower layers. This allows the server to exchange data via various communication media with clients in different AAs. Such a structure would be similar to the structure of a DLMS®/COSEM client show below.

4.1.11 Model of a DLMS®/COSEM client

Figure 8 shows the model of a DLMS®/COSEM client as an example.

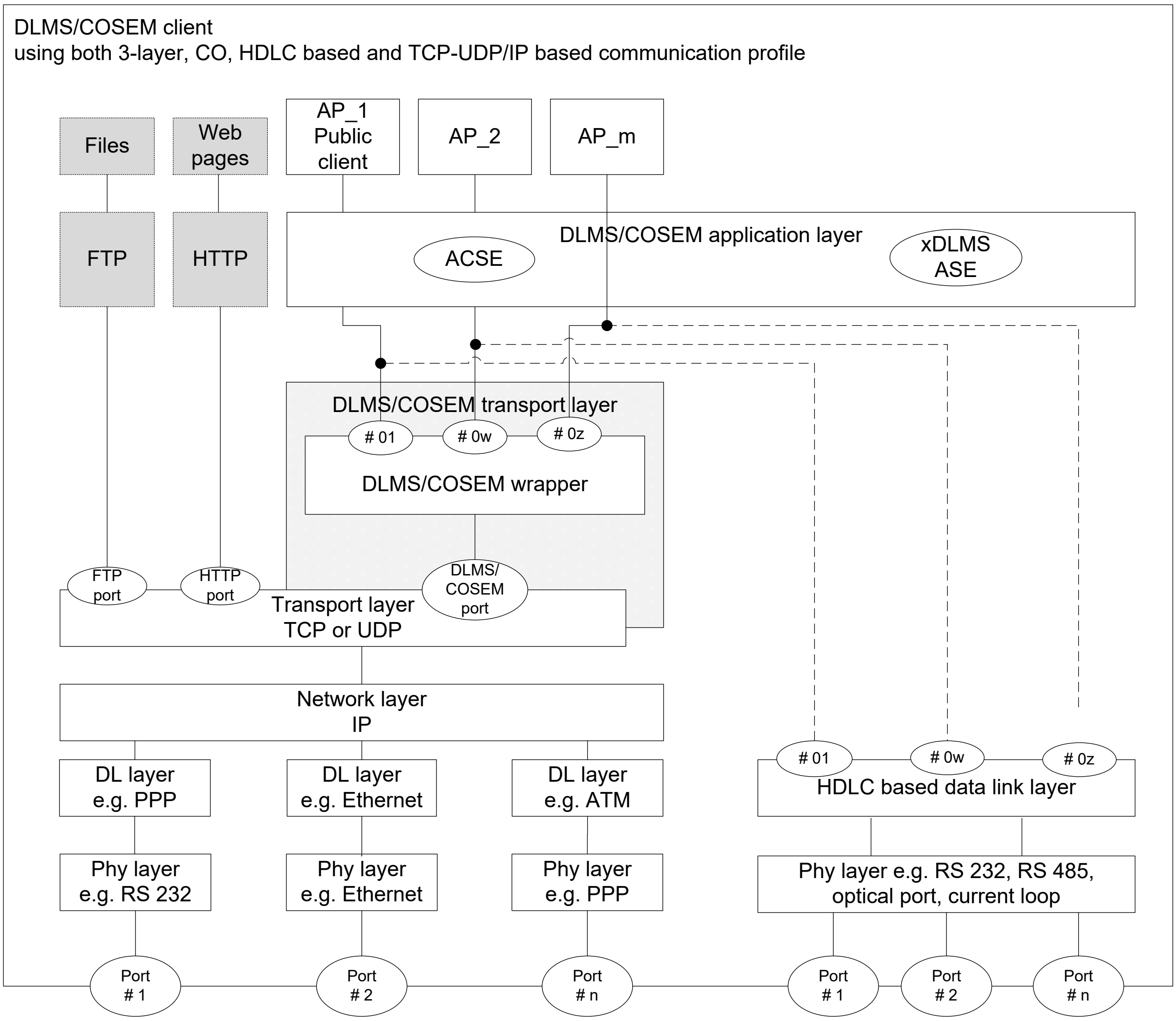


Figure 8 – Model of a DLMS®/COSEM client using multiple protocol stacks

The model of the client – obviously – is very similar to the model of the servers:

- in this particular model, the DLMS®/COSEM AL is supported either by the HDLC based data link layer or the DLMS®/COSEM TL, meaning that the AL uses the services of one or the other as determined by the APs. In other words, the APDUs are received from or sent through the appropriate supporting layer, which in turn use the services of its supporting layer respectively;
- unlike on the server side, the addressing provided by the HDLC layer has a single level only, that of the Service Access Points (SAP) of each Application Process (AP).

As explained, client APs and server APs are identified by their SAPs. Therefore, an AA between a client and a server side AP can be identified by a pair of client and server SAPs.

The DLMS®/COSEM AL may be capable to support one or more AAs simultaneously. Likewise, lower layers may be capable of supporting more than one connection with their peer layers. This allows data exchange between clients and servers simultaneously via different ports and communication media.

4.1.12 Interoperability and interconnectivity in DLMS®/COSEM

In the DLMS®/COSEM environment, interoperability and interconnectivity is defined between client and server AEs. A client and a server AE must be interoperable and interconnectable to ensure data exchange between the two systems.

Using the COSEM object model to model metering of all kinds of energy, over all communication media ensures *semantic interoperability*, i.e. an unambiguous, shared meaning between clients and servers using different communication media. The semantic elements are the COSEM objects, their logical name i.e. the OBIS code, the definition of their attributes and methods and the data types that can be used.

Using the DLMS®/COSEM AL over all communication media ensures *syntactic interoperability*, which is a pre-requisite of *semantic interoperability*. Syntactic interoperability comprises the ability to establish AAs between clients and server using various application contexts, authentication mechanisms, xDLMS contexts and security contexts as well as the standard structure and encoding of all messages exchanged.

Interconnectivity is a protocol level notion: in order to be able to exchange messages, the client and the server AEs should be *interconnectable* and *interconnected*.

Before the two AEs can establish an AA, they must be *interconnected*. The two AEs are interconnected, if each peer protocol layer of both sides, which needs to be connected, is connected. In order to be interconnected, the client and server AEs should be interconnectable and shall establish the required connections. Two AEs are *interconnectable* if they use the same communication profile.

With this, interconnectivity in DLMS®/COSEM is ensured by the ability of the DLMS®/COSEM AE to establish a connection between all peer layers, which need to be connected.

4.1.13 Ensuring interconnectivity: the protocol identification service

In DLMS®/COSEM, AA establishment is always initiated by the client AE. However, in some cases, it may not have knowledge about the protocol stack used by an unknown server device (for example when the server has initiated the physical connection establishment). In such cases, the client AE needs to obtain information about the protocol stack implemented in the server.

A specific, application level service is available for this purpose: the protocol identification service. It is an optional application level service, allowing the client AE to obtain information – after establishing a physical connection – about the protocol stack implemented in the server. The protocol identification service uses directly the data transfer services (PH-DATA.request /.indication) of the PhL; it bypasses the other protocol layers. It is recommended to support it in all communication profiles that have access to the PhL.

4.1.14 System integration and meter installation

System integration is supported by DLMS®/COSEM in a number of ways.

A possible process is described here.

As shown in Figure 6, the presence of a Public Client (bound to address 0x10 in any profile) is mandatory in each client system. Its main role is to reveal the structure of an unknown – for example newly installed – metering equipment. This takes place within a mandatory AA between the Public Client and the Management Logical Device, with no security precautions. Once the structure is known, data can be accessed with using the proper authentication mechanisms and cryptographic protection of the xDLMS messages and COSEM data.

When a new meter is installed in the system, it may generate an event report to the client. Once this is detected, the client can retrieve the internal structure of the meter, and then send the necessary configuration information (for example tariff schedules and installation specific parameters) to the meter. With this, the meter is ready to use.

System integration is also facilitated by the availability of the DLMS®/COSEM conformance testing, described in the Yellow Book, DLMS® UA 1001-1. With this, correct implementation of the specification in metering equipment can be tested and certified.

4.2 DLMS®/COSEM application layer main features

4.2.1 General

This subclause 4.2 provides an overview of the main features provided by the DLMS®/COSEM AL.

4.2.2 DLMS®/COSEM application layer structure

The structure of the client and server DLMS®/COSEM application layers is shown in Figure 9.

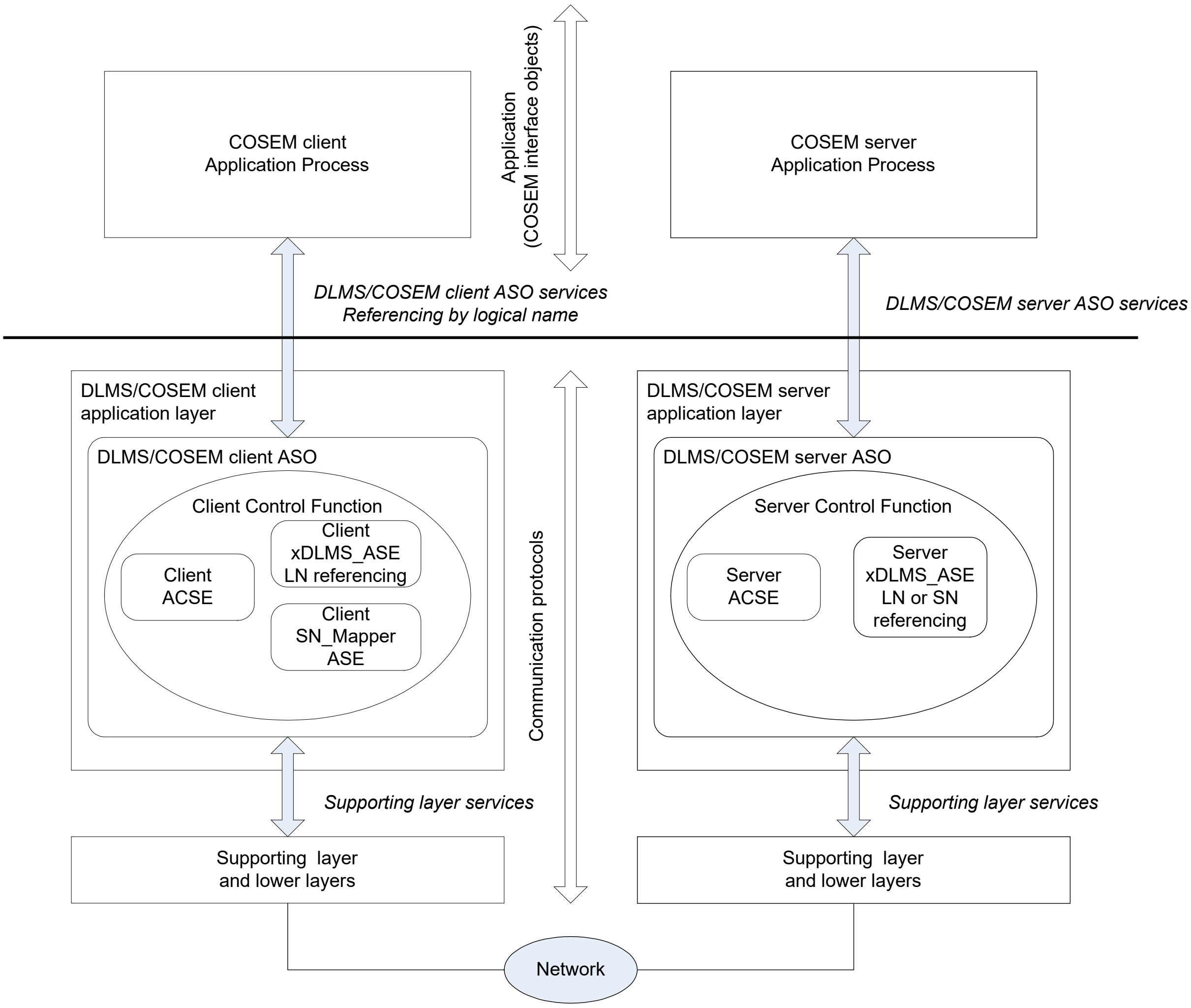


Figure 9 – The structure of the DLMS®/COSEM application layers

The main component of the DLMS®/COSEM AL is the Application Service Object (ASO). It provides services to its service user, the COSEM Application Process (APs) and uses services provided by the supporting layer. It contains three mandatory components both on the client and on the server side:

- the Association Control Service Element, ACSE;
- the extended DLMS® Application Service Element, xDLMS ASE;
- the Control Function, CF.

On the client side, there is a fourth, optional element, called the Client SN_Mapper ASE.

The ACSE provides services to establish and release application associations (AAs). See 4.2.3.

The xDLMS ASE provides services to transport data between COSEM APs. See 4.2.4.

The Control Function (CF) element specifies how the ASO services invoke the appropriate service primitives of the ACSE, the xDLMS ASE and the services of the supporting layer. See also 7.1.

Both the client and the server DLMS®/COSEM ASO may contain other, optional application protocol components.

The optional Client SN_Mapper ASE is present in the client side AL ASO, when the server uses SN referencing. It provides mapping between services using LN and SN referencing. See 4.2.5.

The DLMS®/COSEM AL performs also some functions of the OSI presentation layer:

- encoding and decoding the ACSE APDUs and the xDLMS APDUs, see also 7.2.3;
- alternatively, generating and using XML documents representing ACSE and xDLMS APDUs;
- applying compression and decompression;
- applying, verifying and removing cryptographic protection.

4.2.3 The Association Control Service Element, ACSE

For the purposes of DLMS®/COSEM connection oriented (CO) communication profiles, the CO ACSE, specified in ISO/IEC 15953:1999 and ISO/IEC 15954:1999 is used.

The services provided for application association establishment and release are the following:

- COSEM-OPEN;
- COSEM-RELEASE;
- COSEM-ABORT.

The COSEM-OPEN service is used to establish AAs. It is based on the ACSE A-ASSOCIATE service. It causes the start of use of an AA by those ASE procedures identified by the value of the Application_Context_Name, Security_Mechanism_Name and xDLMS context parameters. AAs may be established in different ways:

- confirmed AAs are established via a message exchange – using the COSEM-OPEN service – between the client and the server to negotiate the contexts. Confirmed AAs can be established between a single client and a single server;
- unconfirmed AAs are established via a message sent – using the COSEM-OPEN service – from the client to the server, using the parameters of the contexts supposed to be supported by the server. Unconfirmed AAs can be established between a client and one or multiple servers;
- pre-established AAs may pre-exist. In this case, the COSEM-OPEN service is not used. The client has to be aware of the contexts supported by the server. A pre-established AA can be confirmed or unconfirmed.

The COSEM-RELEASE service is used to release AAs. If successful, it causes the completion of the use of the AA without loss of information in transit (graceful release). In some communication profiles – for example in the TCP-UDP/IP based profile – the COSEM-RELEASE service is based on the ACSE A-RELEASE service. In some other communication profiles – for example in the 3-layer, CO, HDLC based profile – there is a one-to-one relationship between a confirmed AA and the supporting protocol layer connection. In such profiles AAs can be released simply by disconnecting the corresponding supporting layer connection. Pre-established AAs cannot be released.

The COSEM-ABORT service causes the abnormal release of an AA with the possible loss of information in transit. It does not rely on the ACSE A-ABORT service.

The COSEM-OPEN service is specified in 6.2, the COSEM-RELEASE service in 6.3 and the COSEM-ABORT service in 6.4.

4.2.4 The xDLMS application service element

4.2.4.1 Overview

To access attributes and methods of COSEM objects, the services of the xDLMS ASE are used. It is based on the DLMS® standard, IEC 61334-4-41:1996. This International Standard specifies a range of extensions to extend functionality while maintaining backward compatibility. The extensions comprise the following:

- additional services, see 4.2.4.3;
- additional mechanisms, see 4.2.4.4;
- additional data types, see 4.2.4.5;
- new DLMS® version number, see 4.2.4.6;
- new conformance block, see 4.2.4.7;
- clarification of the meaning of the PDU size, see 4.2.4.8.

4.2.4.2 The xDLMS initiate service

To establish the xDLMS context the xDLMS Initiate service — specified in IEC 61334-4-41:1996, 5.2 — is used. This service is integrated in the COSEM-OPEN service, see 6.2.

4.2.4.3 COSEM object related xDLMS services

4.2.4.3.1 General

COSEM object related xDLMS services are used to access COSEM object attributes and methods.

IEC 62056-6-2:2021 4.1.2 specifies two referencing methods:

- Logical Name (LN) referencing; and
- Short Name (SN) referencing.

For more information on referencing methods, see 4.2.4.4.2.

Therefore, two distinct xDLMS service sets are specified: one exclusively using Logical Name (LN) referencing and the other exclusively using short name (SN) referencing. It can be considered that there are two different xDLMS ASEs: one providing services with LN referencing and the other with SN referencing. The client ASO always uses the xDLMS ASE with LN referencing. The server ASO may use either the xDLMS ASE with LN referencing or the xDLMS ASE with SN referencing or both.

These services may be:

- requested / solicited by the client; or
- unsolicited: these are always initiated by the server without a previous request from the client.

Services requested by the client may be also (see 7.3.2):

- confirmed: in this case, the server provides a response to the request;

- unconfirmed: in this case, the server does not provide a response to the request.

Unsolicited DataNotification from the server may be also (see 9.3.10):

- confirmed: in this case, the client provides a response to acknowledge the receipt of the unsolicited DataNotification
- unconfirmed: in this case, the client does not provide a response to the unsolicited DataNotification.

The additional services – which are not based on DLMS® services specified in IEC 61334-4-41:1996 – are:

- the GET, SET, ACTION and ACCESS used to access COSEM object attributes and methods using LN referencing;
- the DataNotification service used by the server to push data to the client;
- the EventNotification service used by the server to notify the client about events that occur in the server.

4.2.4.3.2 xDLMS services used by the client with LN referencing

In the case of LN referencing, COSEM object attributes and methods are referenced via the identifier of the COSEM object instance to which they belong. For this referencing method, the following additional services are specified:

- the GET service is used by the client to request the server to return the value of one or more attributes, see 6.6;
- the SET service is used by the client to request the server to replace the content of one or more attributes, see 6.7;
- the ACTION service is used by the client to request the server to invoke one or more methods. Invoking methods may imply sending method invocation parameters and receiving return parameters, see 6.8;
- the ACCESS service, a unified service which can be used by the client to access multiple attributes and/or methods with a single .request; see 6.9.

These services can be invoked by the client in a confirmed or unconfirmed manner.

4.2.4.3.3 xDLMS services used by the client with SN referencing

In the case of SN referencing, COSEM object attributes and methods are mapped to DLMS® named variables specified in IEC 61334-4-41:1996, 10.1.2.

The xDLMS services using SN referencing are based on the DLMS® variable access services, specified in IEC 61334-4-41:1996, 10.4 – 10.6 and they are the following:

- the Read service is used by the client to request the server to return the value of one or more attributes or to invoke one or more methods when return parameters are expected. It is a confirmed service. See 6.14;
- the Write service is used by the client to request the server to replace the content of one or more attributes or to invoke one or more methods when no return parameters are expected. It is a confirmed service. See 6.15;
- the UnconfirmedWrite service is used by the client to request the server to replace the content of one or more attributes or to invoke one or more methods when no return parameters are expected. It is an unconfirmed service. See 6.16.

New variants of the Variable_Access_Specification service parameter (see 6.13), the Read.response and the Write.response services have been added to support selective access – see 4.2.4.3.5 – and block transfer, see 4.2.4.4.5.

4.2.4.3.4 Unsolicited services

Unsolicited services are initiated by the server, on pre-defined conditions, e.g. schedules, triggers or events, to inform the client of the value of one or more attributes, as though they had been requested by the client.

To support push operation, the DataNotification service is available, see 6.10. It can be used in application contexts using either SN referencing or LN referencing.

NOTE The DataNotification service is used in conjunction with “Push setup” COSEM objects, see IEC 62056-6-2:2021, 4.4.8.

To support event notification, the following unsolicited services are available:

- with LN referencing, the EventNotification service, see 6.11;
- with SN referencing, the InformationReport service, see 6.17. This service is based on IEC 61334-4-41:1996, 10.7.

4.2.4.3.5 Selective access

In the case of some COSEM interface classes, selective access to attributes is available, meaning that either the whole attribute or a selected portion of it can be accessed as required. For this purpose, access selectors and parameters are specified as part of the specification of the relevant attributes.

To use this possibility, attribute-related services can be invoked with access selection parameters. In the case of LN referencing, this feature is called Selective access; see 6.6 and 6.7. It is a negotiable feature; see 7.3.1. In the case of SN referencing, this feature is called Parameterized access; see 6.14, 6.15 and 6.16. It is a negotiable feature; see 7.3.1.

4.2.4.3.6 Multiple references

In a COSEM object related service invocation, it is possible to reference one or several named variables, attributes and/or methods. Using multiple references is a negotiable feature. See 7.3.1.

4.2.4.3.7 Attribute_0 referencing

With the GET, SET and ACCESS services a special feature, Attribute_0 referencing is available. By convention, attributes of COSEM objects are numbered from 1 to n, where Attribute_1 is the logical name of the COSEM object. Attribute_0 has a special meaning: it references all attributes with positive index (public attributes). The use of Attribute_0 referencing with the GET service is explained in 6.6, with the SET service in 6.7 and with the ACCESS service in 6.9.

NOTE As specified in IEC 62056-6-2:2021, 4.1.2, manufacturers may add proprietary methods and/or attributes to any object, using negative numbers.

Attribute_0 referencing is a negotiable feature, see 7.3.1.

4.2.4.4 Additional mechanisms

4.2.4.4.1 Overview

xDLMS specifies several new mechanisms – compared to DLMS® as specified in IEC 61334-4-41:1996 – to improve functionality, flexibility and efficiency. The additional mechanisms are:

- referencing using logical names;
- identification of service invocations;
- priority handling;
- transferring long application messages;
- composable xDLMS messages;
- compression and decompression;
- general cryptographic protection;
- general block transfer.

4.2.4.4.2 Referencing methods and service mapping

To access COSEM object attributes and methods with the xDLMS services, they have to be referenced. As already mentioned in 4.2.4.3.1, IEC 62056-6-2:2021, 4.1.2 specifies two referencing methods:

- Logical Name (LN) referencing; and
- Short Name (SN) referencing.

In the case of LN referencing, COSEM object attributes and methods are referenced via the logical name (COSEM_Object_Instance_ID) of the COSEM object instance to which they belong. In the case of SN referencing, COSEM object attributes and methods are mapped to DLMS® named variables.

Accordingly, there are two xDLMS ASEs specified: one using xDLMS services with LN referencing and one using xDLMS services with SN referencing.

On the client side, in order to handle the different referencing methods transparently for the AP, the AL uses the xDLMS ASE with LN referencing. Using a unique, standardized service set between COSEM client APs and the communication protocol – hiding the particularities of DLMS®/COSEM servers using different referencing methods – allows specifying an Application Programming Interface, API. This is an explicitly specified interface corresponding to this service set for applications running in a given computing environment (for example Windows, UNIX, etc.) Using this – public – API specification, client applications can be developed without knowledge about particularities of a given server.

On the server side, either the xDLMS ASE with LN referencing or the xDLMS ASE with SN referencing or both xDLMS ASEs can be used.

In the case of confirmed AAs, the referencing method is negotiated during the AA establishment phase via the COSEM application context. It shall not change during the lifetime of the AA established. Using LN or SN services within a given AA is exclusive.

In the case of unconfirmed and pre-established AAs, the client AL is expected to know the referencing method supported by the server.

When the server uses LN referencing, the services are the same on both sides. When the server uses SN referencing the Client SN_Mapper ASE in the client maps the SN referencing into LN referencing or vice versa. See 4.2.2 and 4.2.5.

4.2.4.4.3 Identification of service invocations: the Invoke_Id parameter

In the client/server model, requests are sent by the client and responses are sent by the server. The client is allowed to send several requests before receiving the response to the previous ones, provided that this is allowed by the lower layers.

Therefore – to be able to identify which response corresponds to each request – it is necessary to include a reference in the request.

The Invoke_Id parameter is used for this purpose. The value of this parameter is assigned by the client so that each request carries a different Invoke_Id. The server shall copy the Invoke_Id into the corresponding response.

In the ACCESS and the DataNotification service – see 6.9 and 6.10 – the Long-Invoke-Id parameter is used instead of the Invoke_Id parameter.

The EventNotification service does not contain the Invoke_Id parameter.

This feature is available only with LN referencing.

4.2.4.4.4 Priority handling

For data transfer services using LN referencing, two priority levels are available: normal (FALSE) and high (TRUE). This feature allows receiving a response to a new request before the response to a previous request is completed.

Normally, the server serves incoming service requests in the order of reception (FIFS, First In, First Served). However, a request with the priority parameter set to high (TRUE) is served before the previous requests with priority set to normal (FALSE). The response carries the same priority flag as that of the corresponding request. Managing priority is a negotiable feature; see 7.3.1.

NOTE 1 As service invocations are identified with an Invoke_Id, services with the same priority can be served in any order.

NOTE 2 If the feature is not supported, requests with HIGH priority are served with NORMAL priority.

This feature is not available with services using SN referencing. The server treats the services on a FIFS basis.

4.2.4.4.5 Transferring long messages

The xDLMS service primitives are carried in an encoded form by xDLMS APDUs. This encoded form may be longer than the Client / Server Max Receive PDU Size negotiated. To transfer such 'long' messages, there are two mechanisms available:

a) the general block transfer (GBT) mechanism specified in 4.2.4.4.9;
b) service-specific block transfer mechanism. This mechanism is available with the GET, SET, ACTION, Read and Write services. In this case, the service primitive invocations contain only one part – one block – of the data (e.g. attribute values), so that the encoded form fits in a single APDU.

   NOTE There is no block-recovery mechanism with the service-specific block transfer mechanism.

Using the general or the service-specific block transfer mechanism is a negotiable feature, see 7.3.1.

An APDU that fits in the Client / Server Max Receive PDU Size negotiated may be too long to fit in a single frame / packet of the supporting layer. Such APDUs may be transported if the supporting layer provide(s) segmentation; see Annex A.

4.2.4.4.6 Composable xDLMS messages

The three important aspects of dealing with xDLMS messages are encoding / decoding, applying, verifying / removing cryptographic protection and block transfer.

The concept of composable xDLMS messages separates the three aspects, as shown in Figure 10. See also Figure 36.

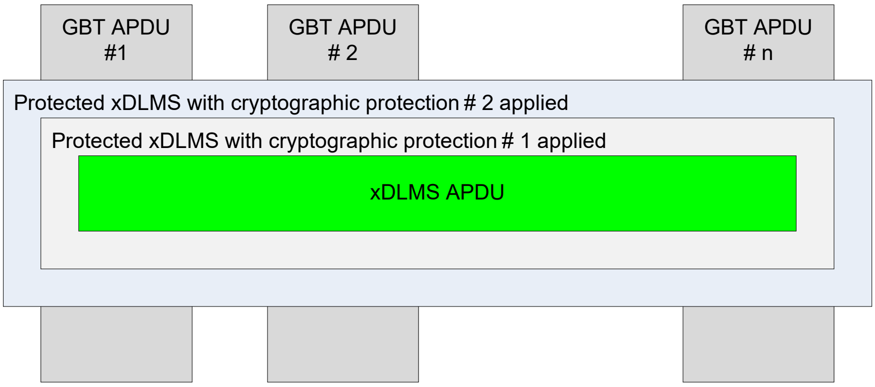


Figure 10 – The concept of composable xDLMS messages

Once the APDU corresponding to the service primitive invoked by the AP is built by the AL, the general protection mechanism can be used to apply cryptographic protection. When an unprotected or a protected APDU is too long to fit in the negotiated APDU size, then the general block transfer mechanism can be applied.

These mechanisms can be applied with all xDLMS APDUs.

NOTE 1 With the GET, SET, ACTION, EventNotification, Read and Write, UnconfirmedWrite and InformationReport APDUs, service-specific cryptographic protection is available using specific service protection types and APDUs.

NOTE 2 With the GET, SET, ACTION, Read, Write, and UnconfirmedWrite and APDUs, service-specific block transfer is available using specific service request / response types and APDUs.

4.2.4.4.7 Compression and decompression

In order to optimize the use of communication media, it is possible to compress xDLMS APDUs to be sent and decompress xDLMS APDUs received. For details, see 5.3.6.

4.2.4.4.8 General protection

This mechanism can be used to apply cryptographic protection to any xDLMS APDU and this allows applying multiple layers of protection between the client and the server or between a third party and the server. See also 5.2.5

For this purpose, the following APDUs are available; see 5.7.2.3:

- the general-ded-ciphering and the general-glo-ciphering APDUs;
- the general-ciphering APDUs;
- the general-signing APDU.

Using the general protection mechanism is a negotiable feature, see 7.3.1.

4.2.4.4.9 General block transfer (GBT)

This mechanism can be used to transfer any xDLMS APDU in blocks. With GBT, the blocks are carried by general-block-transfer APDUs instead of service-specific “with-datablock” APDUs.

NOTE 1 The ACCESS and the DataNotification services do not provide a service-specific block transfer mechanism.

The GBT mechanism supports bi-directional block transfer, streaming and lost block recovery:

- bi-directional block transfer means that while one party is sending blocks, the other party not only confirms the blocks received but if it has blocks to send it can send them as well while it is still receiving blocks;

  NOTE 2 Bi-directional block transfer is useful when long service parameters need to be transported in both directions.

- streaming means that several blocks may be sent – streamed – by one party without an acknowledgement of each block from the other party;
- lost block recovery means that if the reception of a block is not confirmed, it can be sent again. If streaming is used, lost block recovery takes place at the end of the streaming window.

The GBT mechanism is managed by the AL using the block transfer streaming parameters specified in 8 and 9.2.

Using the general block transfer mechanism is a negotiable feature, see 7.3.1.

The protocol of the general block transfer mechanism is specified in 7.3.13

4.2.4.5 Additional data types

The additional data types are specified in Clause 8 and in Clause 9.

4.2.4.6 xDLMS version number

The new DLMS® version number, corresponding to the first version of the xDLMS ASE is 6.

4.2.4.7 xDLMS conformance block

The xDLMS conformance block enables optimised DLMS®/COSEM server implementations with extended functionality. It can be distinguished from the DLMS® conformance block by its tag "Application 31". See 7.3.1, Clause 8 and Clause 9.

The xDLMS conformance block is part of the xDLMS context.

In the case of confirmed AAs, the conformance block is negotiated during the AA establishment phase via the xDLMS context. It shall not change during the lifetime of the AA established.

In the case of unconfirmed and pre-established AAs, the client AL is expected to know the conformance block supported by the server.

4.2.4.8 Maximum PDU size

To clarify the meaning of the maximum PDU size usable by the client and the server, the modifications shown in Table 2 have been made. The xDLMS Initiate service uses these names for PDU sizes.

Table 2 – Clarification of the meaning of PDU size for DLMS®/COSEM

was:	**new:**
IEC 61334-4-41:1996, 5.2.2, Table 3	
Proposed Max PDU Size	Client Max Receive PDU Size
Negotiated Max PDU Size	Server Max Receive PDU Size
IEC 61334-4-41:1996, 5.2.3, 7th paragraph	
The Proposed Max PDU Size parameter, of type Unsigned16, proposes a maximum length expressed in bytes for the exchanged DLMS® APDUs. The value proposed in an Initiate request shall be large enough to always permit the Initiate Error PDU transmission	The Client Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS® APDU that the server may send. The client will discard any received PDUs that are longer than this maximum length. The value shall be large enough to always permit the AARE APDU transmission. Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.
IEC 61334-4-41:1996, 5.2.3, last paragraph	
The Negotiated Max PDU Size parameter, of type Unsigned16, contains a maximum length expressed in bytes for the exchanged DLMS® APDUs. A PDU that is longer than this maximum length will be discarded. This maximum length is computed as the minimum of the Proposed Max PDU Size and the maximum PDU size than the VDE-handler may support.	The Server Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS® APDU that the client may send. The server will discard any received PDUs that are longer than this maximum length. Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.

4.2.5 Layer management services

Layer management services have local importance only. Therefore, specification of these services is not within the Scope of this International Standard.

The specific SetMapperTable service is defined in 6.18.

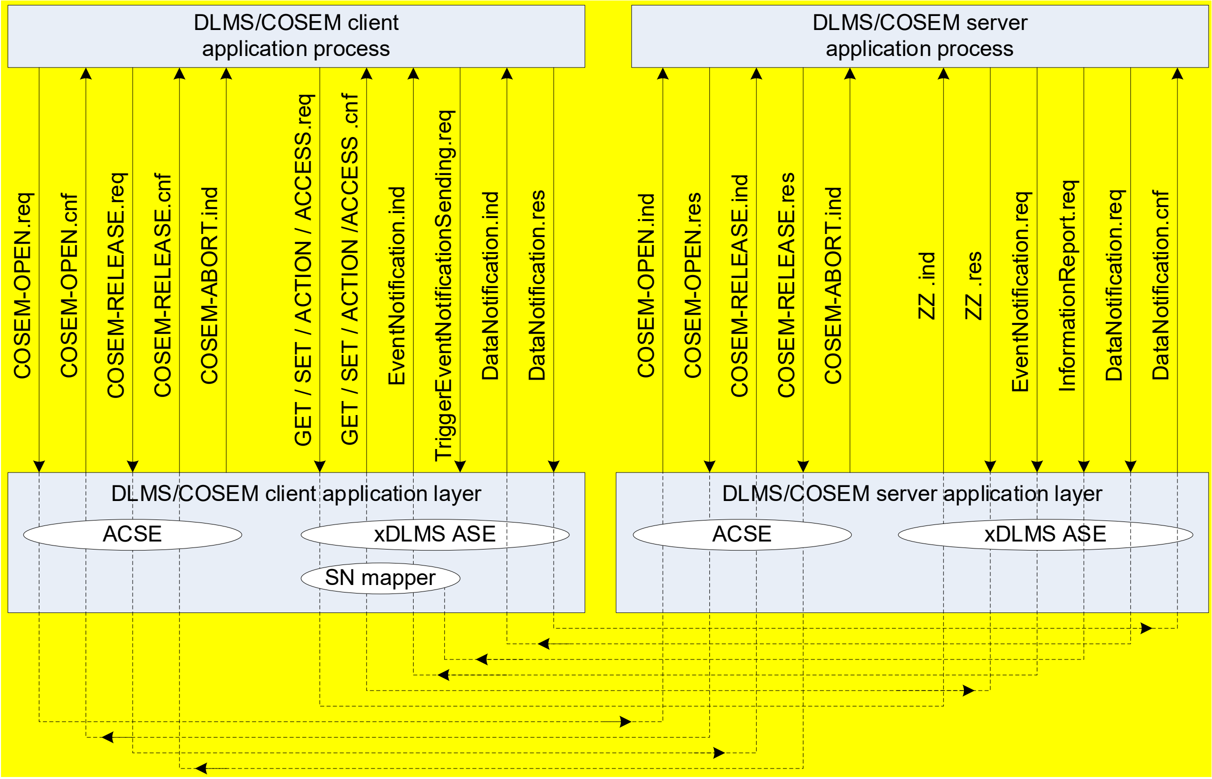
4.2.6 Summary of DLMS®/COSEM application layer services

A summary of the services available at the top of the DLMS®/COSEM AL is shown in Figure 11. Layer management services are not shown. Although the service primitives are different on the client and server side, the APDUs are the same.

NOTE 1 For example, when the client AP invokes a GET.request service primitive the client AL builds a GET-Request APDU. When this is received by the server AL, it invokes a GET.ind service primitive.

The DLMS®/COSEM AL services are specified in 6. The DLMS®/COSEM AL protocol is specified in 7. The abstract syntax of the ACSE and xDLMS APDUs is specified in 7.3.13. The XML schema is defined in 9.

Encoding examples are provided in Annex D, Annex E, and Annex F.



NOTE 2 The client AP always uses LN referencing. If the server uses SN referencing then a mapping is performed by the Client SN_Mapper ASE. Consequently, the service primitives ZZ.ind and ZZ.res may be LN or SN service primitives. LN/SN service mapping is specified in 9.5.

NOTE 3 The ACCESS service cannot be mapped to services using SN referencing.

Figure 11 – Summary of DLMS®/COSEM AL services

4.2.7 DLMS®/COSEM application layer protocols

The DLMS®/COSEM AL protocols specify the procedures for information transfer for AA control and authentication using connection-oriented ACSE procedures, and for data transfer between COSEM clients and servers using xDLMS procedures. Therefore, the DLMS®/COSEM AL protocol is based on the ACSE standard as specified in ISO/IEC 15954:1999 and the DLMS® standard, as specified in IEC 61334-4-41:1996, with the extensions for DLMS®/COSEM. The procedures are defined in terms of:

- the interactions between peer ACSE and xDLMS protocol machines through the use of services of the supporting protocol layer;
- the interactions between the ACSE and xDLMS protocol machines and their service user.

The DLMS®/COSEM AL protocols are specified in 7.

5 Information security in DLMS®/COSEM

5.1 Overview

This Clause 5 describes and specifies:

- the DLMS®/COSEM security concept, see 5.2;
- the cryptographic algorithms selected, see 5.3;
- the security keys, see 5.4, 5.5 and 5.6;
- the use of the cryptographic algorithms for entity authentication, xDLMS APDU protection and COSEM data protection, see 5.7.

5.2 The DLMS®/COSEM security concept

5.2.1 Overview

The resources of DLMS®/COSEM servers – COSEM object attributes and methods – can be accessed by DLMS®/COSEM clients within Application Associations, see also 4.1.5.

During an AA establishment the client and the server have to identify themselves. The server may also require that the *user* of a client identifies itself. Furthermore, the server may require that the client authenticates itself and the client may also require that the server authenticates itself. The identification and authentication mechanisms are specified in 5.2.2.

Once an AA is established, xDLMS services can be used to access COSEM object attributes and methods, subject to the security context and access rights. See 5.2.3 and 5.2.4.

The xDLMS APDUs carrying the services primitives can be cryptographically protected. The required protection is determined by the security context and the access rights. To support end-to-end security between third parties and servers, such third parties can also access the resources of a server using a client as a broker. The concept of message protection is further explained in 5.2.5.

Moreover, COSEM data carried by the xDLMS APDUs can be cryptographically protected; see 5.2.6.

As these security mechanisms are applied on the application process / application layer level, they can be used in all DLMS®/COSEM communication profiles.

NOTE Lower layers may provide additional security.

5.2.2 Identification and authentication

5.2.2.1 Identification

As specified in 4.1.3.3, DLMS®/COSEM AEs are bound to Service Access Points (SAPs) in the protocol layer supporting the AL. These SAPs are present in the PDUs carrying the xDLMS APDUs within an AA.

The client user identification mechanism enables the server to distinguish between different users on the client side— that may be operators or third parties — to log their activities accessing the meter. See also 4.1.3.6.

5.2.2.2 Authentication mechanisms

5.2.2.2.1 Overview

The authentication mechanisms determine the protocol to be used by the communication entities to authenticate themselves during AA establishment. There are three different authentication mechanisms available with different authentication security levels:

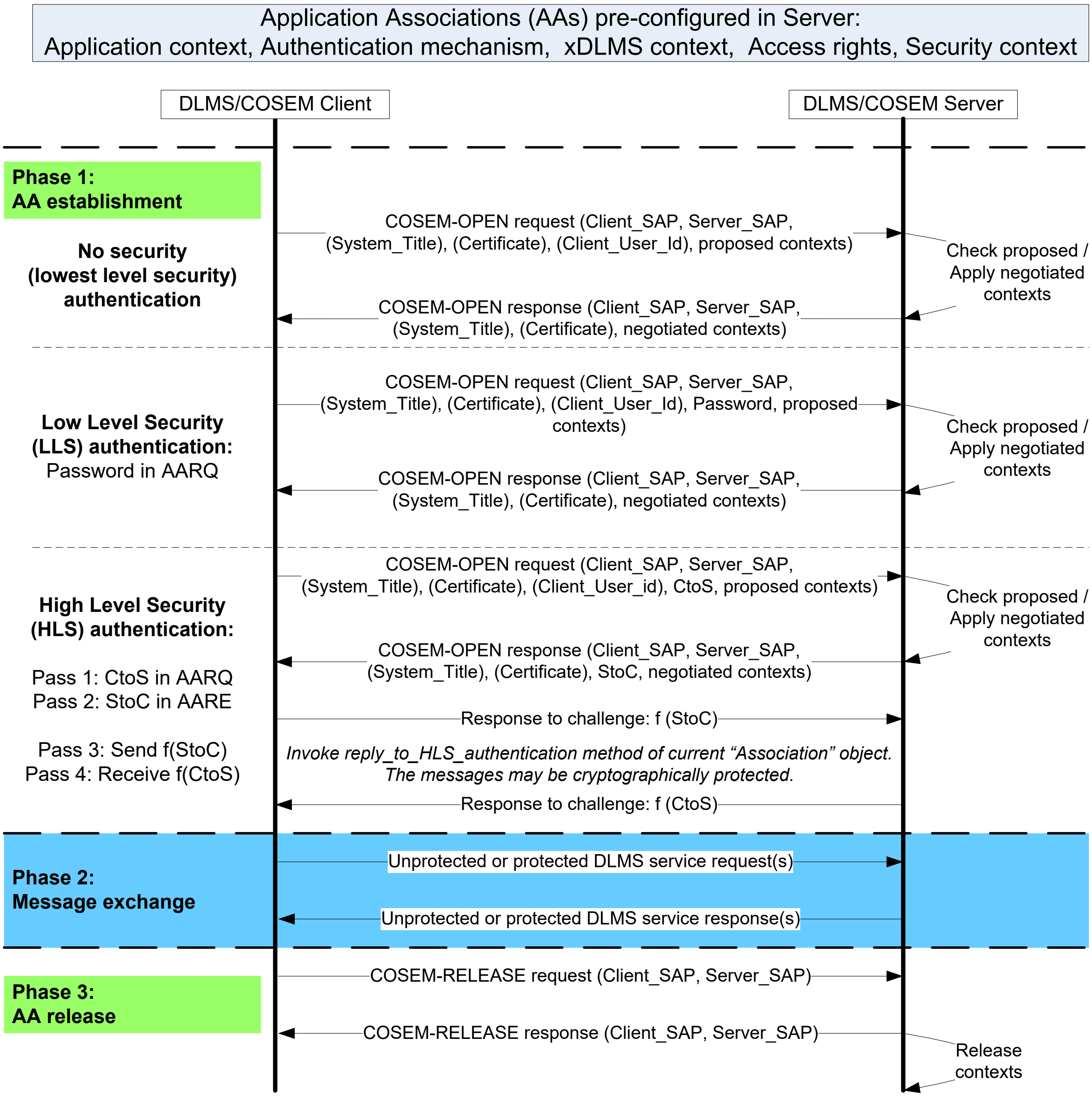
- no security (Lowest Level Security) authentication; see 5.2.2.2.2;
- Low Level Security (LLS) authentication, see 5.2.2.2.3;

  NOTE 1 In ITU-T X.811 this is known as unilateral authentication, class 0 mechanism.

- High Level Security (HLS) authentication, see 5.2.2.2.4.

  NOTE 2 In ITU-T X.811 this is known as mutual authentication using challenge mechanisms.

They are shown in Figure 12. Authentication mechanisms are identified by names, see 7.2.2.3.



NOTE 1 The COSEM-OPEN service primitives are carried by AARQ / AARE APDUs. The COSEM-RELEASE service primitives are carried by RLRQ / RLRE APDUs (when used). See 6.2 and 6.3.

NOTE 2 The elements (*System_Title), (Certificate) and (Client_User_Id)* are optional.

NOTE 3 In pre-established AAs no authentication takes place.

NOTE 4 The COSEM-RELEASE service can be cryptographically protected by including a ciphered xDLMS Initiate .request / .response APDU in the RLRQ.

Figure 12 – Authentication mechanisms

The security of the message exchange (in Phase 2) is independent of the client-server authentication during AA establishment (Phase 1). Even in the case where no client-server authentication takes place, cryptographically protected APDUs can be used to ensure message security.

5.2.2.2.2 No security (Lowest Level Security) authentication

The purpose of No security (Lowest Level Security) authentication is to allow the client to retrieve some basic information from the server. This authentication mechanism does not require any authentication; the client can access the COSEM object attributes and methods within the security context and access rights prevailing in the given AA.

5.2.2.2.3 Low Level Security (LLS) authentication

In this case, the server requires that the client authenticates itself by supplying a password that is known by the server. The password is held by the current "Association SN / LN" object modelling the AA to be established. The "Association SN / LN" objects provide means to change the secret.

If the password supplied is accepted, the AA can be established, otherwise it shall be rejected.

LLS authentication is supported by the COSEM-OPEN service – see 6.2 – as follows:

- the client transmits a "secret" (a password) to the server, using the COSEM-OPEN.request service primitive;
- the server checks if the "secret" is correct;
- if yes, the client is authenticated and the AA can be established. From this moment, the negotiated contexts are valid;
- if not, the AA shall be rejected;
- the result of establishing the AA shall be sent back by the server using the COSEM-OPEN.response service primitive, together with diagnostic information.

5.2.2.2.4 High Level Security (HLS) authentication

In this case, both the client and the server have to successfully authenticate themselves to establish an AA. HLS authentication is a four-pass process that is supported by the COSEM-OPEN service and the *reply_to_HLS_authentication* method of the "Association SN / LN" interface class:

- Pass 1: The client transmits a "challenge" $CtoS$ and – depending on the authentication mechanism – additional information to the server;
- Pass 2: The server transmits a "challenge" $StoC$ and – depending on the authentication mechanism – additional information to the client;

If $StoC$ is the same as $CtoS$, the client shall reject it and shall abort the AA establishment process.

- Pass 3: The client processes $StoC$ and the additional information according to the rules of the HLS authentication mechanism valid for the given AA and sends the result to the server. The server checks if f($StoC$) is the result of correct processing and – if so – it accepts the authentication of the client;
- Pass 4: The server processes then $CtoS$ and the additional information according to the rules of the HLS authentication mechanism valid for the given AA and sends the result to the client. The client checks if f($CtoS$) is the result of correct processing and – if so – it accepts the authentication of the server.

Pass 1 and Pass 2 are supported by the COSEM-OPEN service.

After Pass 2 – provided that the proposed application context and xDLMS context are acceptable – the server grants access to the method *reply_to_HLS_authentication* of the current "Association SN / LN" object using the application context negotiated.

Pass 3 and Pass 4 are supported by the method *reply_to_HLS_authentication* of the "Association SN / LN" object(s). If both passes 3 and 4 are successfully executed, then the AA is established with the application context and xDLMS context negotiated.

The dedicated-key, if transferred, can be used from this moment.

Otherwise, either the client or the server aborts.

There are several HLS authentication mechanisms available. These are further specified in 5.7.4.

In some HLS authentication mechanisms, the processing of the challenges involves the use of an HLS secret.

The "Association SN / LN" interface class provides a method to change the HLS "secret": *change_HLS_secret*.

REMARK After the client has issued the change_HLS_secret () – or change_LLS_secret () – method, it expects a response from the server acknowledging that the secret has been changed. It is possible that the server transmits the acknowledgement, but due to communication problems, the acknowledgement is not received at the client side. Therefore, the client does not know if the secret has been changed or not. For simplicity reasons, the server does not offer any special support for this case; i.e. it is left to the client to cope with this situation.

5.2.3 Security context

The security context defines security attributes relevant for cryptographic transformations and includes the following elements:

- the security suite, determining the security algorithms available, see 5.3.7;
- the security policy, determining the kind(s) of protection to be applied generally to all xDLMS APDUs exchanged within an AA. The possible security policies are specified in 5.7.2.2;
- the security material, relevant for the given security algorithms, that includes security keys, initialization vectors, public key certificates and the like. As the security material is specific for each security algorithm, the elements are specified in detail in the relevant clauses.

The security context is managed by "Security setup" objects; see IIEC 62056-6-2:2021, 4.4.7.

5.2.4 Access rights

Access rights to attributes may be: *no_access*, *read_only*, *write_only*, or *read_and_write*. Access rights to methods may be *no_access* or *access*.

In addition, access rights may stipulate cryptographic protection to be applied to xDLMS APDUs carrying the service primitives used to access a particular COSEM object attribute / method. The protection required on the .request and on the .response can be independently configured.

Access rights are held by the relevant "Association SN / LN" objects; see IEC 62056-6-2:2021, 4.4.3 and 4.4.4. The possible access rights are specified in 5.7.2.2.

The protection to be applied shall meet the stronger of the requirement stipulated by the security policy and the access rights.

5.2.5 Application layer message security

DLMS®/COSEM ensures AL level security by providing means to cryptographically protect xDLMS APDUs. The protection may be any combination of authentication, encryption and digital signature and can be applied in a multi-layer fashion by multiple parties. The protection is applied by the originator and is verified and removed by the recipient.

A request or response received shall be processed only if the protection on the message carrying the request or response could be successfully verified and removed.

Project specific companion specifications may specify additional criteria for accepting and processing messages.

The concept of message protection between a client and a server is shown in Figure 13.

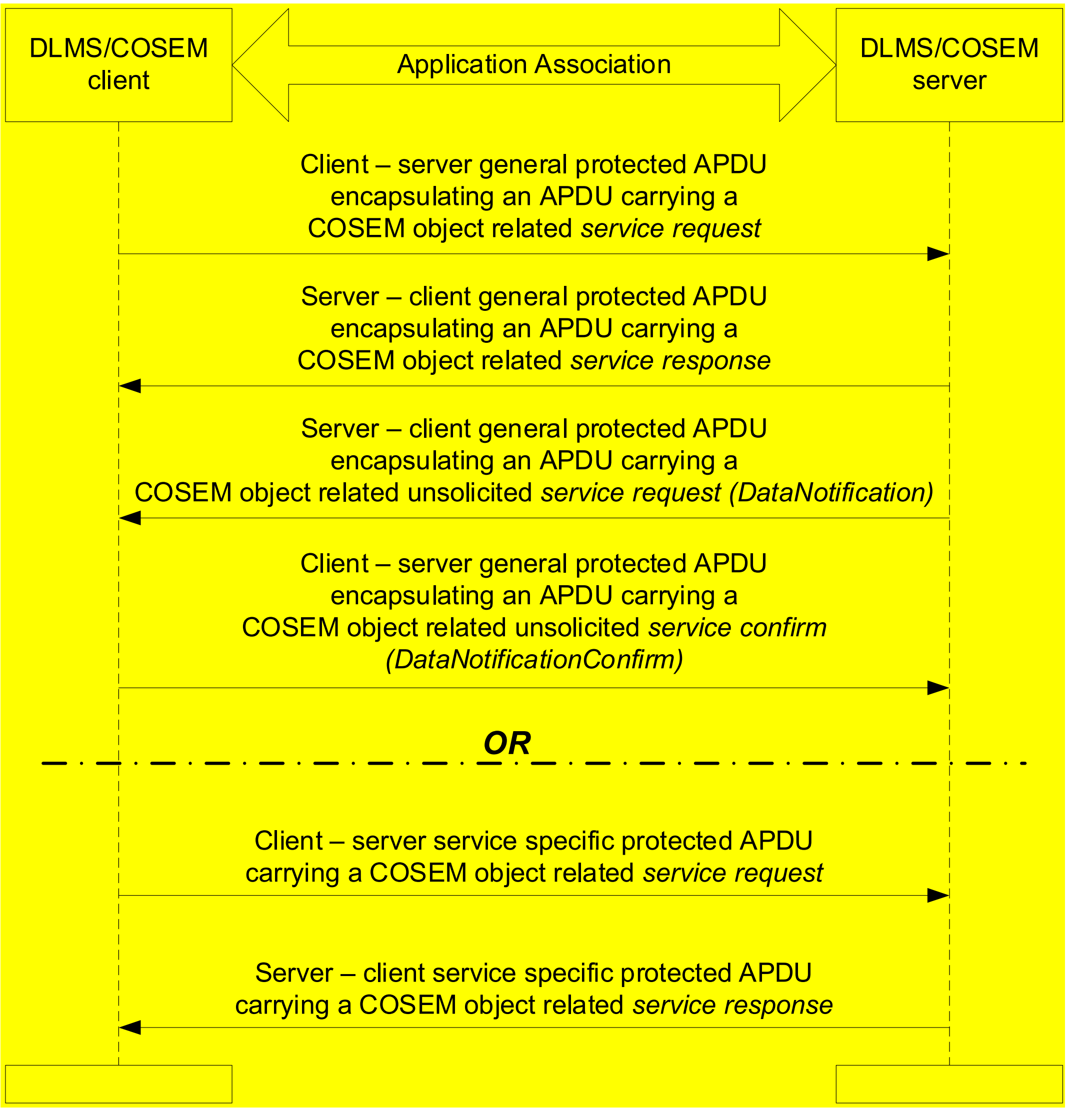


Figure 13 – Client – server message security concept

To ensure end-to-end message security, third parties have to be able to exchange protected xDLMS service requests with DLMS®/COSEM servers. In this case, the client acts as a broker, meaning that a third party is a user of one of the AAs between a client and a server the third party wants to reach.

The concept of message protection between a third party and a server is shown in Figure 14.

The third party:

- is DLMS®/COSEM aware i.e. it can generate and process messages encapsulating xDLMS APDUs carrying COSEM object related service requests and responses;
- it is able to apply its own protection to the xDLMS APDU carrying the request;
- it is able to verify protection applied by the server and / or the client on the response.

The DLMS®/COSEM client:

- acts as a broker between the third party and the server;
- makes an appropriate AA available for use by the third party, based on information included in the TP – client message;
- verifies that the TP has the right to use that AA;

  NOTE 2 The way to verify this is outside the Scope of this International Standard.

- it may verify the protection applied by the third party;
- encapsulates the third party – client message into a general protected xDLMS APDU;
- it may verify the protection applied by the server on the APDU encapsulating the COSEM object related service response or unsolicited service request; (in the case of Push operation);
- it may apply its own protection to the protected xDLMS APDUs sent to the TP.

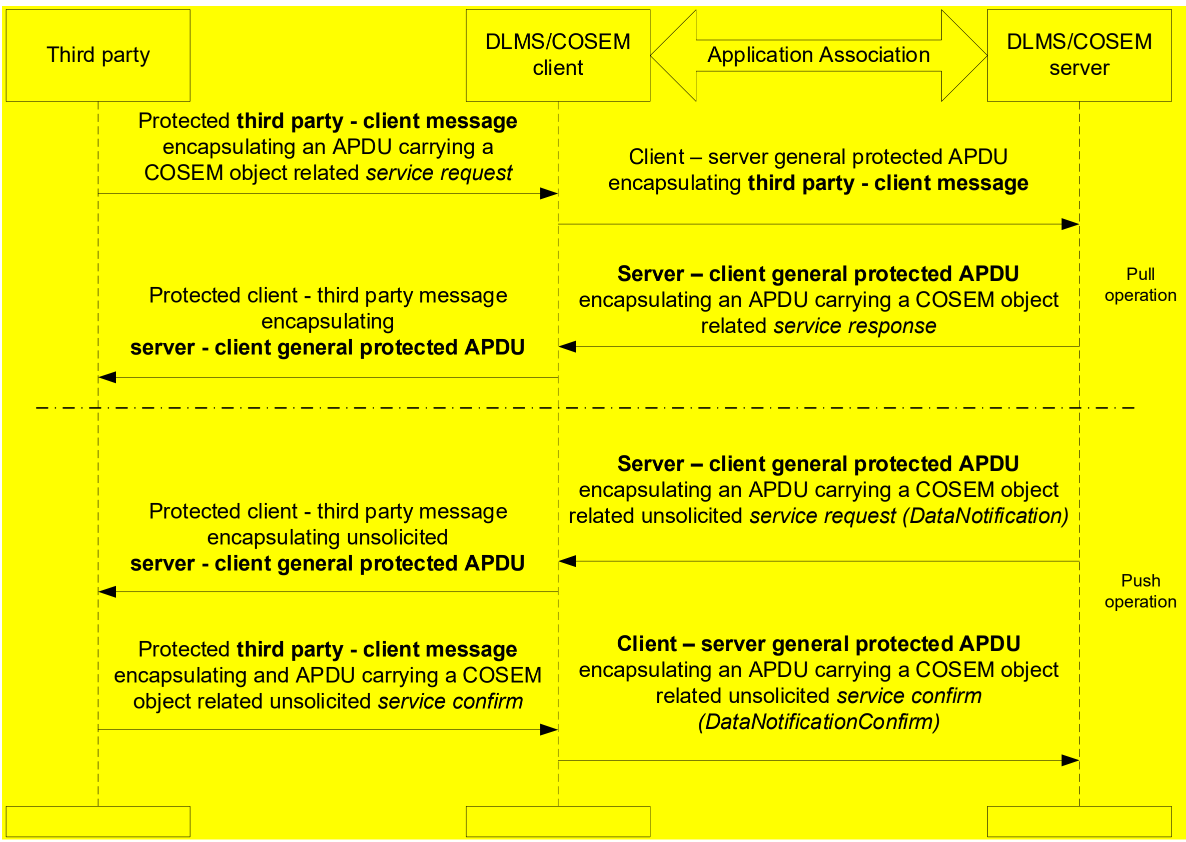


Figure 14 – End-to-end message security concept

The server:

- shall (pre-)establish an AA with the client used by the third party;
- it may check the identity of the third party using the AA;

- it shall provide access to COSEM object attributes and methods as determined by the security policy and access rights once the protection(s) applied by the client and/or the third party have been successfully verified;
- it shall prepare the response – or, in the case of Push operation an unsolicited service request – and apply the protection determined by the protection applied on the incoming request, the access rights and the security policy.

The application of cryptographic protection on xDLMS APDUs is specified in 5.7.2.

5.2.6 COSEM data security

COSEM data i.e. values of COSEM object attributes, method invocation parameters and return parameters can be also cryptographically protected. When this is required, the attributes and methods concerned are accessed indirectly, via “Data protection” objects, that apply and verify / remove protection on COSEM data; see IEC 62056-6-2:2021, 4.4.9.

See also 5.7.5.

5.3 Cryptographic algorithms

5.3.1 Overview

DLMS®/COSEM applies cryptography to protect the information.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.1.

Cryptography is a branch of mathematics that is based on the transformation of data and can be used to provide several security services: confidentiality, data integrity, authentication, authorization and non-repudiation. Cryptography relies upon two basic components: an *algorithm* (or cryptographic methodology) and a *key*. The algorithm is a mathematical function, and the key is a parameter used in the transformation.

A cryptographic algorithm and key are used to apply cryptographic protection to data (e.g., encrypt the data or generate a digital signature) and to remove or check the protection (e.g., decrypt the encrypted data or verify the digital signature). There are three basic types of approved cryptographic algorithms:

- cryptographic hash functions that do not require keys (although they can be used in a mode in which keys are used). A hash function is often used as a component of an algorithm to provide a security service. See 5.3.2;
- symmetric key algorithms (often called secret key algorithms) that use a single key – shared by a sender and a receiver – to both apply the protection and to remove or check the protection. Symmetric key algorithms are relatively easy to implement and provide a high throughput. See 5.3.3;
- asymmetric key algorithms (often called public key algorithms) that use two keys (i.e., a key pair): a public key and a private key that are mathematically related to each other. Compared to symmetric key algorithms, implementation of asymmetric key algorithms is complex and requires much more computation. See 5.3.4.

In order to use cryptography, cryptographic keys must be “in place”, i.e., keys must be established for parties using cryptography. See 5.4.

5.3.2 Hash function

NOTE The following text is quoted from NIST SP 800-21:2005, 3.2.

A hash function produces a short representation of a longer message. A good hash function is a one-way function: it is easy to compute the hash value from a particular input; however, backing up the process from the hash value back to the input is extremely difficult. With a good hash function, it is also extremely difficult to find two specific inputs that produce the same hash value. Because of these characteristics, hash functions are often used to determine whether or not data has changed.

A hash function takes an input of arbitrary length and outputs a fixed length value. Common names for the output of a hash function include *hash value* and *message digest*. Figure 15 depicts the use of a hash function.

A hash value (H1) is computed on data (M1). M1 and H1 are then saved or transmitted. At a later time, the correctness of the retrieved or received data is checked by labelling the received data as M2 (rather than M1) and computing a new hash value (H2) on the received value. If the newly computed hash value (H2) is equal to the retrieved or received hash value (H1), then it can be assumed that the retrieved or received data (M2) is the same as the original data (M1) (i.e., M1 = M2).

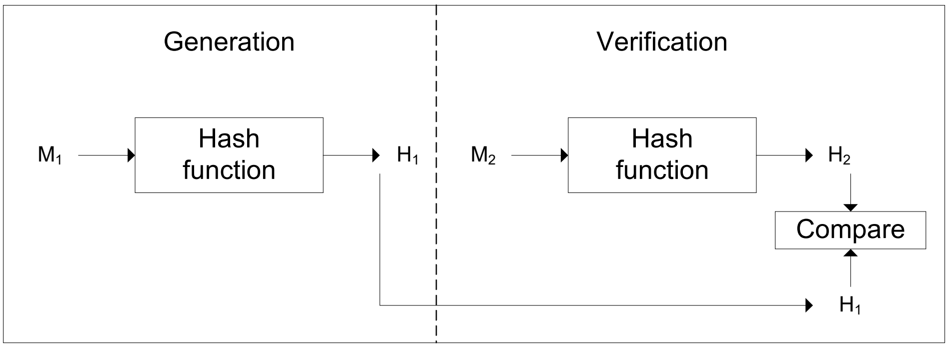


Figure 15 – Hash function

Hash algorithms are used in DLMS®/COSEM for the following purposes:

- digital signature, see 5.3.4.4;
- key agreement, see 5.3.4.6; and
- HLS authentication. The algorithm to be used depends on the authentication mechanism, see 5.7.4.

For digital signature and key agreement the algorithm shall be as stipulated by the security suite, see Table 9.

5.3.3 Symmetric key algorithms

5.3.3.1 General

Symmetric key algorithms are used in DLMS®/COSEM for the following purposes:

- authentication of communicating partners using HLS authentication mechanisms, see 5.7.4;
- authentication and encryption of xDLMS messages, see 5.7.2;
- authentication and encryption of COSEM data, see 5.7.5.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.

Symmetric key algorithms (often called secret key algorithms) use a single key to both apply the protection and to remove or check the protection. For example, the key used to encrypt data is also used to decrypt the encrypted data. This key must be kept secret if the data is to retain its cryptographic protection. Symmetric key algorithms are used to provide confidentiality via encryption, or an assurance of authenticity or integrity via authentication, or are used during key establishment.

Keys used for one purpose shall not be used for other purposes. (See NIST SP 800-57:2012).

5.3.3.2 Encryption and decryption

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.

Encryption is used to provide confidentiality for data. The data to be protected is called *plaintext*. Encryption transforms the data into *ciphertext*. Ciphertext can be transformed back into plaintext using decryption.

Plaintext data can be recovered from ciphertext only by using the same key that was used to encrypt the data. Unauthorized recipients of the ciphertext who know the cryptographic algorithm but do not have the correct key should not be able to decrypt the ciphertext. However, anyone who has the key and the cryptographic algorithm can easily decrypt the ciphertext and obtain the original plaintext data.

Figure 16 depicts the encryption and decryption processes. The plaintext (P) and a key (K) are used by the encryption process to produce the ciphertext (C). To decrypt, the ciphertext (C) and the same key (K) are used by the decryption process to recover the plaintext (P).

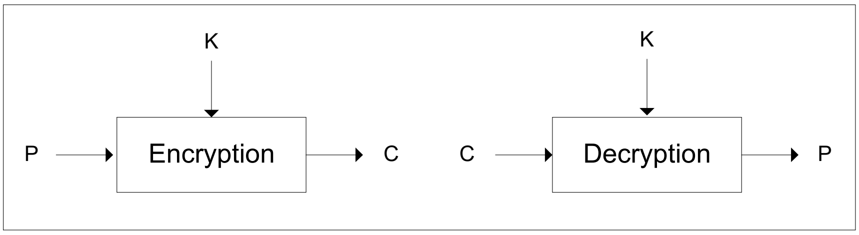


Figure 16 – Encryption and decryption

With symmetric key block cipher algorithms, the same plaintext block and key will always produce the same ciphertext block. This property does not provide acceptable security. Therefore, cryptographic modes of operation have been defined to address this problem (see 5.3.3.4).

5.3.3.3 Advanced Encryption Standard

For the purposes of DLMS®/COSEM, the Advanced Encryption Standard (AES) as specified in FIPS PUB 197:2001 shall be used. AES operates on blocks (chunks) of data during an encryption or decryption operation. For this reason, AES is referred to as a block cipher algorithm.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.3.

AES encrypts and decrypts data in 128-bit blocks, using 128, 192 or 256 bit keys. All three key sizes are adequate.

AES offers a combination of security, performance, efficiency, ease of implementation, and flexibility. Specifically, the algorithm performs well in both hardware and software across a wide range of computing environments. Also, the very low memory requirements of the algorithm make it very well suited for restricted-space environments.

5.3.3.4 Encryption Modes of Operation

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.4.

With a symmetric key block cipher algorithm, the same plaintext block will always encrypt to the same ciphertext block when the same symmetric key is used. If the multiple blocks in a typical message (data stream) are encrypted separately, an adversary could easily substitute individual blocks, possibly without detection. Furthermore, certain kinds of data patterns in the plaintext, such as repeated blocks, would be apparent in the ciphertext.

Cryptographic modes of operation have been defined to address this problem by combining the basic cryptographic algorithm with variable initialization values (commonly known as initialization vectors) and feedback rules for the information derived from the cryptographic operation.

NIST SP 800-38D:2007 specifies the Galois/Counter Mode (GCM), an algorithm for authenticated encryption with associated data, and its specialization, GMAC, for generating a message authentication code (MAC) on data that is not encrypted. GCM and GMAC are modes of operation for an underlying approved symmetric key block cipher. See 5.3.3.3.

5.3.3.5 Message Authentication Code

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.2.

Message Authentication Codes (MACs) provide an assurance of authenticity and integrity. A MAC is a cryptographic checksum on the data that is used to provide assurance that the data has not changed or been altered and that the MAC was computed by the expected party (the sender). Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

Figure 17 depicts the use of message authentication codes (MACs).

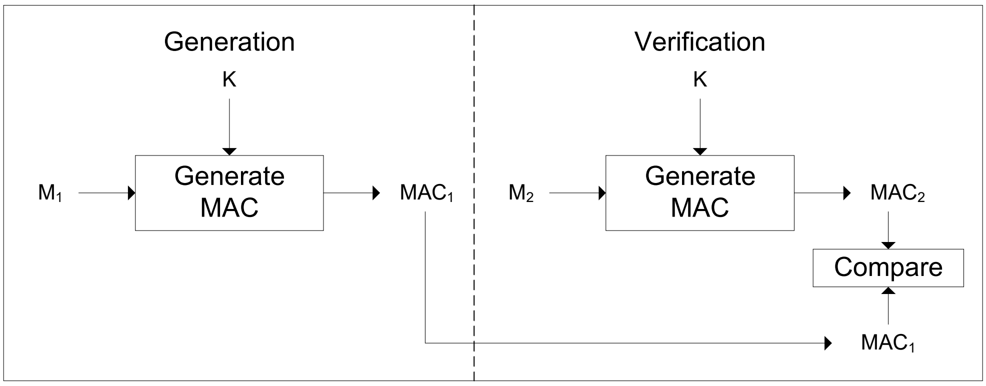


Figure 17 – Message Authentication Codes (MACs)

A MAC (MAC1) is computed on data (M1) using a key (K). M1 and MAC1 are then saved or transmitted. At a later time, the authenticity of the retrieved or received data is checked by labelling the retrieved or received data as M2 and computing a MAC (MAC2) on it using the same key (K). If the retrieved or received MAC (MAC1) is the same as the newly computed MAC (MAC2), then it can be assumed that the retrieved or received data (M2) is the same as the original data (M1) (i.e., M1 = M2). The verifying party also knows who the sending party is because no one else knows the key.

Typically, MACs are used to detect data modifications that occur between the initial generation of the MAC and the verification of the received MAC. They do not detect errors that occur before the MAC is originally generated.

Message integrity is frequently provided using non-cryptographic techniques known as error detection codes. However, these codes can be altered by an adversary to the adversary's benefit. The use of an approved cryptographic mechanism, such as a MAC, addresses this problem. That is, the integrity provided by a MAC is based on the assumption that it is not possible to generate a MAC without knowing the cryptographic key. An adversary without knowledge of the key will be unable to modify data and then generate an authentic MAC on the modified data. It is therefore crucial that MAC keys be kept secret.

For the purposes of DLMS®/COSEM, the GMAC algorithm as specified in 5.3.3.7.2 shall be used.

5.3.3.6 Key wrapping

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.3.

Symmetric key algorithms may be used to wrap (i.e., encrypt) keying material using a key-wrapping key (also known as a key encrypting key). The wrapped keying material can then be stored or transmitted securely. Unwrapping the keying material requires the use of the same key-wrapping key that was used during the original wrapping process.

Key wrapping differs from simple encryption in that the wrapping process includes an integrity feature. During the unwrapping process, this integrity feature detects accidental or intentional modifications to the wrapped keying material. For the purposes of DLMS®/COSEM the AES key wrap algorithm shall be used; see 5.3.3.8.

5.3.3.7 Galois/Counter Mode

5.3.3.7.1 General

NOTE The following text is taken from NIST SP 800-38D:2007, Clause 3.

Galois/Counter Mode (GCM) is an algorithm for authenticated encryption with associated data. GCM is constructed from an approved symmetric key block cipher with a block size of 128 bits, such as the Advanced Encryption Standard (AES) algorithm, see FIPS PUB 197. Thus, GCM is a mode of operation of the AES algorithm.

GCM provides assurance of the confidentiality of data using a variation of the Counter mode of operation for encryption.

GCM provides assurance of the authenticity of the confidential data (up to about 64 gigabytes per invocation) using a universal hash function that is defined over a binary Galois (i.e., finite) field (GHASH). GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted.

If the GCM input is restricted to data that is not to be encrypted, the resulting specialization of GCM, called GMAC, is simply an authentication mode on the input data.

GCM provides stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both 1) accidental modifications of the data and 2) intentional, unauthorized modifications.

In DLMS®/COSEM, it is also possible to use GCM to provide confidentiality only: in this case, the authentication tags are simply not computed and checked.

5.3.3.7.2 GCM functions

NOTE The following is based on NIST SP 800-38D:2007, 5.2.

The two functions that comprise GCM are called authenticated encryption and authenticated decryption; see Figure 18.

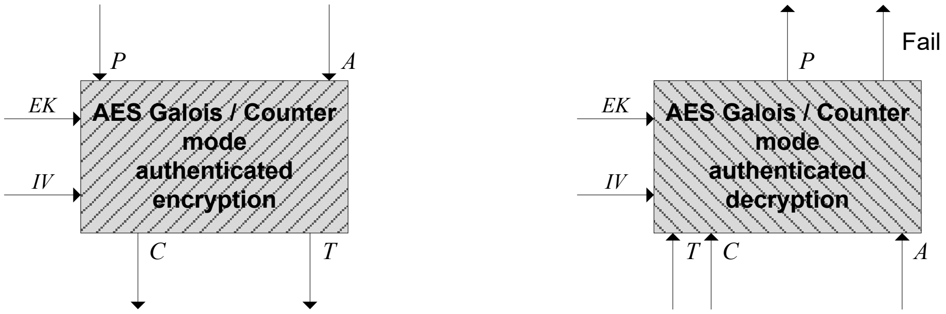


Figure 18 – GCM functions

The authenticated encryption function encrypts the confidential data and computes an authentication tag on both the confidential data and any additional, non-confidential data. The authenticated decryption function decrypts the confidential data, contingent on the verification of the tag.

When the input is restricted to non-confidential data, the resulting variant of GCM is called GMAC. For GMAC, the authenticated encryption and decryption functions become the functions for generating and verifying an authentication tag on the non-confidential data.

Finally, if authentication is not required, the authenticated encryption function encrypts the confidential data, but the authentication tag is not computed. The authenticated decryption function decrypts the confidential data, but no authentication tag is computed and verified.

In DLMS®/COSEM, the use of authentication and encryption is indicated by bit 4 and bit 5 of the Security Control Byte, specified in 5.7.2.4.

a) The authenticated encryption function – given the selection of a block cipher key EK – has three input strings:

- a plaintext, denoted P;
- Additional Authenticated Data (AAD), denoted A;
- an initialization vector (IV) denoted IV.

The plaintext and the AAD are the two categories of data that GCM protects. GCM protects the authenticity of the plaintext and the AAD; GCM also protects the confidentiality of the plaintext, while the AAD is left in the clear.

The IV is essentially a nonce, i.e. a value that is unique within the specified (security) context, which determines an invocation of the authenticated encryption function on the input data to be protected. See 5.3.3.7.3.

The bit lengths of the input strings to the authenticated encryption function shall meet the following requirements:

- len(P) < 2^{39}-256;
- len(A) < 2^{64}-1;
- 1 ≤ len(IV) ≤ 2^{64}-1.

The bit lengths of P, A and IV shall all be multiples of 8, so that these values are byte strings.

There are two outputs:

- a ciphertext, denoted C whose bit length is the same as that of the plaintext P;
- an authentication tag, or tag, for short, denoted T.

b) The authenticated decryption function – given the selection of a block cipher key EK – has four input strings:

- the initialization vector, denoted IV;
- the ciphertext, denoted C;
- the Additional Authenticated Data (AAD), denoted A;
- the authentication tag, denoted T.

The output is one of the following:

- the plaintext P that corresponds to the ciphertext C, or
- a special error code denoted $FAIL$ in this International Standard.

The output P indicates that T is the correct authentication tag for IV, A, and C; otherwise, the output is $FAIL$.

5.3.3.7.3 The initialization vector, *IV*

In DLMS®/COSEM, for the construction of the initialization vector IV deterministic construction as specified in NIST SP 800-38D:2007, 8.2.1 shall be used: the IV is the concatenation of two fields, called the fixed field and the invocation field. The fixed field shall identify the physical device, or, more generally, the (security) context for the instance of the authenticated encryption function. The invocation field shall identify the sets of inputs to the authenticated encryption function in that particular device.

For any given key, no two distinct physical devices shall share the same fixed field, and no two distinct sets of inputs to any single device shall share the same invocation field.

The length of the IV shall be 96 bits (12 octets): $\mathrm{len}(IV) = 96$. Within this:

- the leading (i.e. the leftmost) 64 bits (8 octets) shall hold the fixed field. It shall contain the system title, see 4.1.3.4;
- the trailing (i.e. the rightmost) 32 bits shall hold the invocation field. The invocation field shall be an integer counter.

Each encryption key (*EK*) has two invocation counters (*IC*) associated with it, one for the authenticated encryption function and the other for the authenticated decryption function. The *EK* is for block cyphering. The following rules apply:

- when the key is established, the corresponding *IC's* are reset to 0;
- when the authenticated encryption function is used, the corresponding *IC* is used after which it is incremented by 1. If the maximum value of the *IC* has been reached, any further invocations of the authenticated encryption function shall return an error and the *IC* shall not be incremented.
- When the authenticated decryption function is used, the value of the *IC* is verified. The value must be equal to or greater than the lowest acceptable value.
  If the value being verified satisfies this requirement, the lowest acceptable is set to the *IC* value verified plus 1 following the use of the authentication decryption function.
  If the value being verified is less than the lowest acceptable value, the verification fails and so does the authenticated decryption function.
  If the value being verified is equal to the maximum value, the authenticated decryption function shall return an error.

NOTE The maximimum number of invocations is 2^{32}-1.

The bit length of the fixed field limits the number of distinct physical devices that can implement the authenticated encryption function for the given key to 2^{64}. The bit length of the invocation field limits the number of invocations of the authenticated encryption function to 2^{32} with any given input sets without violating the uniqueness requirement.

5.3.3.7.4 The encryption key, *EK*

GCM uses a single key, the block cipher key. In DLMS®/COSEM, this is known as the encryption key, denoted *EK*. Its size depends on the security suite – see 5.3.7 – and shall be:

- for security suite 0 and 1, 128 bits (16 octets): $\mathrm{len}(EK)$ = 128;
- for security suite 2, 256 bits (32 octets): $\mathrm{len}(EK)$ = 256;

The key shall be generated uniformly at random, or close to uniformly at random, i.e., so that each possible key is (nearly) equally likely to be generated. Consequently, the key will be fresh, i.e., unequal to any previous key, with high probability. The key shall be secret and shall be used exclusively for GCM with the chosen block cipher AES. Additional requirements on the establishment and management of keys are discussed in NIST SP 800-38D:2007, 8.1.

5.3.3.7.5 The authentication key, *AK*

In DLMS®/COSEM, for additional security, an authentication key denoted *AK* is also specified. When present, it shall be part of the Additional Authenticated Data, AAD. For its length and its generation, the same rules apply as for the encryption key.

5.3.3.7.6 Length of the authentication tag

The bit length of the authentication tag, denoted t, is a security parameter. In security suites 0, 1 and 2 its value shall be 96 bits.

5.3.3.8 AES key wrap

For wrapping key data DLMS®/COSEM has selected the AES key wrap algorithm specified in RFC 3394. The algorithm is designed to wrap or encrypt key data. It operates on blocks of 64 bits. Before being wrapped, the key data is parsed into *n* blocks of 64 bits. The only restriction the key wrap algorithm places on *n* is that *n* has to be at least two.

The AES key wrap can be configured to use any of the three key sizes supported by the AES codebook: 128, 192, 256.

The two algorithms are key wrap and key unwrap.

The inputs to the key wrapping process are the Key Encrypting Key *KEK* and the plaintext to be wrapped. The plaintext consists of n 64-bit blocks, containing the key data being wrapped. The output is the ciphertext, (n+1) 64 bit values.

The inputs to the unwrap process are the *KEK* and (n+1) 64-bit blocks of ciphertext consisting of previously wrapped key. It returns n blocks of plaintext consisting of the n 64-bit blocks of the decrypted key data.

In DLMS®/COSEM, the size of *KEK* depends on the security suite – see 5.3.7 – and shall be:

- for security suite 0 and 1, 128 bits (16 octets): $\mathrm{len}(KEK)$ = 128;
- for security suite 2, 256 bits (32 octets): $\mathrm{len}(KEK)$ = 256.

5.3.4 Public key algorithms

5.3.4.1 General

In general, public key cryptography systems use hard-to-solve problems as the basis of the algorithm. The RSA algorithm is based on the prime factorization of very large integers. Elliptic Curve Cryptography (ECC) is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECC provides similar levels of security compared to RSA but with significantly reduced key sizes. ECC is particularly suitable for embedded devices and therefore it has been selected for use in DLMS®/COSEM.

Public key algorithms are used in DLMS®/COSEM for the following purposes:

- authentication of communicating partners;
- digital signature of xDLMS APDUs and COSEM data;
- key agreement.

NOTE 1 The following text is quoted from NIST SP 800-21:2005, 3.4.

Asymmetric key algorithms (often called public key algorithms) use two keys: a public key and a private key, which are mathematically related to each other. The public key may be made public; the private key must remain secret if the data is to retain its cryptographic protection. Even though there is a relationship between the two keys, the private key cannot be determined from the public key. Which key to be used to apply versus remove or check the protection depends on the service to be provided. For example, a digital signature is computed using a private key and the signature is verified using the public key; for those algorithms also capable of encryption, the encryption is performed using the public key, and the decryption is performed using the private key.

NOTE 2 Not all public key algorithms are capable of multiple functions, e.g., generating digital signatures and encryption. Asymmetric key algorithms are not used for encryption in DLMS®/COSEM.

Asymmetric key algorithms are used primarily as data integrity, authentication, and non-repudiation mechanisms (i.e., digital signatures), as well as for key establishment.

Some asymmetric key algorithms use domain parameters, which are additional values necessary for the operation of the cryptographic algorithm. These values are mathematically related to each other. Domain parameters are usually public and are used by a community of users for a substantial period of time.

The secure use of asymmetric key algorithms requires that users obtain certain assurances:

- assurance of domain parameter validity provides confidence that the domain parameters are mathematically correct;
- assurance of public key validity provides confidence that the public key appears to be a suitable key; and
- assurance of private key possession provides confidence that the party that is supposedly the owner of the private key really has the key.

Some asymmetric key algorithms may be used for multiple purposes (e.g., for both digital signatures and key establishment). Keys used for one purpose shall not be used for other purposes.

5.3.4.2 Elliptic curve cryptography

5.3.4.2.1 General

Elliptic curve cryptography involves arithmetic operations on an elliptic curve over a finite field. Elliptic curves can be defined over any field of numbers (i.e., real, integer, complex) although they are most often used over finite prime fields for applications in cryptography.

An elliptic curve on a prime field consists of the set of real numbers (x, y) that satisfy the equation:

$$y^2 = x^3 + ax + b$$

The set of all of the solutions to the equation forms the elliptic curve. Changing a and b changes the shape of the curve, and small changes in these parameters can result in major changes in the set of (x, y) solutions.

5.3.4.2.2 NIST recommended elliptic curves

FIPS PUB 186-4:2013 recommends five prime field elliptic curves over a prime field $GF(p)$. Of these, the curves P-256 and P-384 have been selected for DLMS®/COSEM as shown in Table 3.

Table 3 – Elliptic curves in DLMS®/COSEM security suites

DLMS® security suite	Curve name in FIPS PUB 186-4:2013	ASN.1 Object Identifier
Suite 0	–	–
Suite 1	NIST curve P-256	1.2.840.10045.3.1.7
Suite 2	NIST curve P-384	1.3.132.0.34
NOTE The ASN.1 Object Identifier appears in the Certificate under AlgorithmIdentifier: Parameters. See 5.6.4.2.		

5.3.4.3 Data conversions

5.3.4.3.1 Overview

This clause describes the data conversion primitives that shall be used to convert between different data types used to specify public key algorithms: octet strings (OS), bit strings (BS), integers (I), field elements (FE) and elliptic curve points (ECP). DLMS®/COSEM uses octet strings to represent elements of public key algorithms and uses conversion primitives between these data types from and to octet strings. The octet string $M_{d-1}\ M_{d-2}\ \ldots\ M_0$ of length d is encoded as A-XDR OCTET STRING where the leftmost octet M_{d-1} corresponds to first octet of the encoded value of the OCTET STRING.

5.3.4.3.2 Conversion between Bit Strings and Octet Strings (BS2OS)

The data conversion primitive that converts a bit string to an octet string is called the Bit String to Octet String Conversion Primitive, or BS2OS. It takes the bit string as input and outputs the octet string. The bit string $b_{l-1}\ b_{l-2}\ \ldots\ b_0$ of length l shall be converted to an octet string $M_{d-1}\ M_{d-2}\ \ldots\ M_0$ of length $d = \lceil l/8 \rceil$.

The conversion pads enough zeroes on the left to make the number of bits multiple of eight, and then breaks it into octets.

More precisely, conversion shall be as follows:

- for $0 \le i < d - 1$, let the octet $M_i = b_{8i+7}\ b_{8i+6} \ldots b_{8i}$,;
- the leftmost octet M_{d-1} shall have its leftmost $8d - l$ bits set to zero;
- its rightmost $8 - (8d - l)$ bits shall be $b_{l-1}\ b_{l-2} \ldots b_{8d-8}$.

5.3.4.3.3 Conversion between Octet Strings and Bit Strings (OS2BS)

The data conversion primitive that converts an octet string to a bit string is called the Octet String to Bit String Conversion Primitive, or OS2BS. It takes the octet string as input and outputs the bit string. The octet string $M_{d-1}\ M_{d-2} \ldots M_0$ of length d shall be converted to a bit string $b_{l-1}\ b_{l-2} \ldots b_0$ of desired length l. where $d = \lceil l/8 \rceil$ and the leftmost $8d\text{-}l$ bits of the leftmost octet are zero.

More precisely conversion shall be as follows:

- for $0 \le i < l - 1$, let the bits $b_{8i+7}\ b_{8i+6} \ldots b_{8i} = M_i$;
- its leftmost $(8d - l)$ bits of the leftmost octet shall be zero.

5.3.4.3.4 Conversion between Integers and Octet Strings (I2OS)

The data conversion primitive that converts an integer to an octet string is called the Integer to Octet String Conversion Primitive, or I2OS. It takes a non-negative integer x and the desired length d of the octet string as input. The length d has to satisfy $256^d > x$, otherwise it shall output “error”. I2OS outputs the corresponding octet string.

The integer x shall be written in its unique l-digit representation base 256:

- $x = x^{d-1} \cdot 256^{d-1} + x^{d-2} \cdot 256^{d-2} + \cdots + x_1 \cdot 256 + x_0$,;
- where $0 \le x_i < 256$ for $0 \le i \le d\text{-}1$;
- $M_i = x_i$, for $0 \le i \le d\text{-}1$.

The output octet string shall be $M_{d-1}\ M_{d-2} \ldots M_0$.

5.3.4.3.5 Conversion between Octet Strings and Integers (OS2I)

The data conversion primitive that converts an octet string to an integer is called the Octet String to Integer Conversion Primitive, or OS2I. It takes the octet string $M_{d-1}\ M_{d-2} \ldots M_0$ of length d as an input and outputs the corresponding integer x. In the case of the octet string of length zero, the conversion outputs integer 0.

Each octet is interpreted as a non-negative integer to the base 256. More precisely, conversion shall be as follows:

- $x_i = M_i$, for $0 \le i \le d\text{-}1$;
- $x = x_{d-1} \cdot 256^{d-1} + x_{d-2} \cdot 256^{d-2} + \cdots + x_1 \cdot 256 + x_0$.

5.3.4.3.6 Conversion between Field Elements and Octet Strings (FE2OS)

The data conversion primitive that converts a field element to an octet string is called the Field Element to Octet String Conversion Primitive, or FE2OS. It takes a field element as input and outputs the corresponding octet string. A field element $x \in F_p$ is converted to an octet string $M_{d-1}\ M_{d-2} \ldots M_0$ of length $d = \lceil log256\ p \rceil$ by applying I2OS conversion primitive with parameter l, where

- FE2OS(x) = I2OS(x, l).

5.3.4.3.7 Conversion between Octet Strings and Field Elements (OS2FE)

The data conversion primitive that converts an octet string to a field element is called the Octet String to Field Element Conversion Primitive, or OS2FE. It takes an octet string as input and outputs the corresponding field element. An octet string $M_{d-1}\, M_{d-2}\, \dots\, M_0$ of length d is converted to field element $x \in F_p$ by applying OS2I conversion primitive where:

- OS2FE(x) = OS2I(x) mod p.

5.3.4.4 Digital signature

NOTE The following text is quoted from NIST SP 800-21:2005, 3.4.1.

A digital signature is an electronic analogue of a written signature that can be used in proving to the recipient or a third party that the message was signed by the originator (a property known as non-repudiation). Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at a later time.

Digital signatures authenticate the integrity of the signed data and the identity of the signatory. A digital signature is represented in a computer as a string of bits and is computed using a digital signature algorithm that provides the capability to generate and verify signatures. Signature generation uses a private key to generate a digital signature. Signature verification uses the public key that corresponds to, but is not the same as, the private key to verify the signature. Each signatory possesses a private and public key pair. Signature generation can be performed only by the possessor of the signatory's private key. However, anyone can verify the signature by employing the signatory's public key. The security of a digital signature system is dependent on maintaining the secrecy of a signatory’s private key. Therefore, users must guard against the unauthorized acquisition of their private keys.

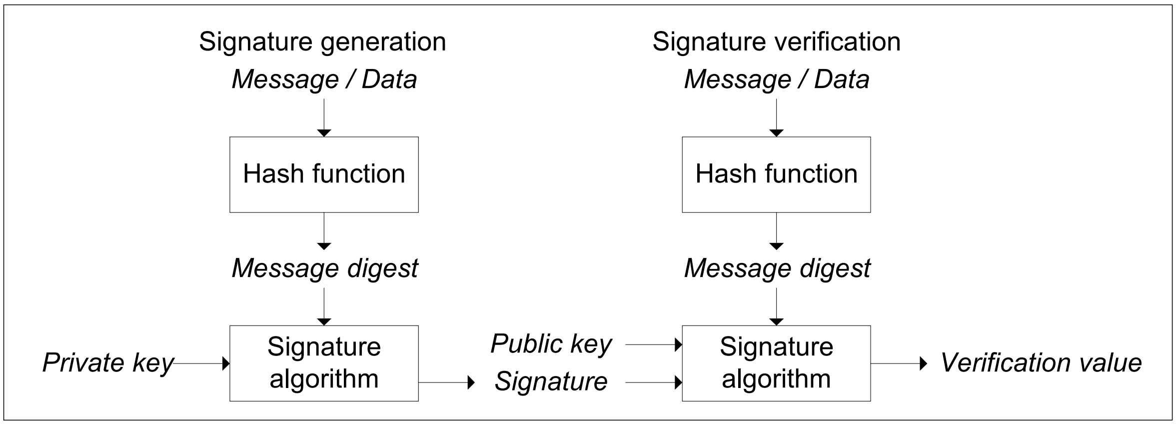


Figure 19 – Digital signatures

Figure 19 depicts the digital signature process. A hash function (see 5.3.2) is used in the signature generation process to obtain a condensed version of data to be signed, called a message digest or hash value. The message digest is then input to the digital signature algorithm to generate the digital signature. The digital signature is sent to the intended verifier along with the signed data (often called the message). The verifier of the message and signature verifies the signature by using the signatory's public key. The same hash function and digital signature algorithm must also be used in the verification process. Similar procedures may be used to generate and verify signatures for stored as well as transmitted data.

5.3.4.5 Elliptic curve digital signature (ECDSA)

For DLMS®/COSEM the elliptic curve digital signature (ECDSA) algorithm as specified in FIPS PUB 186-4:2013 has been selected. NSA1 provides an implementation guide.

In DLMS®/COSEM the elliptic curves and algorithms used shall be:

- in the case of Security Suite 1, the elliptic curve P-256 with the SHA-256 hash algorithm;
- in the case of Security Suite 2, the elliptic curve P-384 with the SHA-384 hash algorithm.

The inputs to ECDSA digital signature generation are the following:

- the message M to be signed;

  NOTE 1 In the DLMS®/COSEM context "Message" may be an xDLMS APDU, see 5.7.2 or COSEM data, see 5.7.5.

- the private key of the signatory, d.

The output is the ECDSA signature (r, s) over M.

In DLMS®/COSEM the plain format shall be used: the signature (r, s) is encoded as octet string R II S, i.e. as concatenation of the octet strings R = I2OS(r, l) and S = I2OS(s, l) with $l = [log_{256} n]$. Thus, the signature has a fixed length of 2 l octets.

NOTE 2 Here, n is the order of the base point G of the elliptic curve. I2OS is the Integer to Octet String Conversion Primitive. See 5.3.4.3.

The inputs to the verification of the ECDSA digital signature generation are the following:

- the signed message M';
- the received ECDSA signature (r', s');
- the authentic public key of the signatory, Q.

The process of generating and verifying the signatures shall be as specified in NSA1, 3.4.

5.3.4.6 Key agreement

5.3.4.6.1 Overview

Key agreement allows two entities to jointly compute a shared secret and derive secret keying material from it. See also NIST SP 800-56A Rev. 2: 2013.

For DLMS®/COSEM three elliptic curve key agreement schemes have been selected from NIST SP 800-56A Rev. 2: 2013:

- the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme;
- the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme;
- the Static Unified Model C(0e, 2s, ECC CDH) scheme.

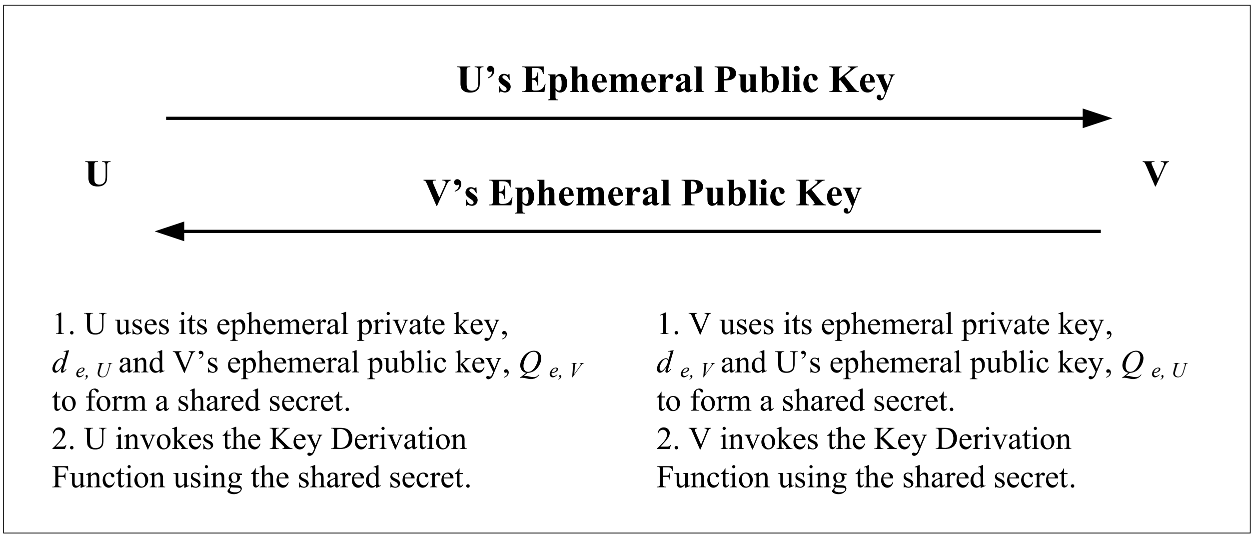
NOTE For NSA Suite B the first two schemes have been selected.

5.3.4.6.2 The Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

This scheme is for use between a DLMS®/COSEM client and a server to agree on the master key, on global encryption keys and/or on the authentication key. The client plays the role of party U and the server plays the role of party V. The process is supported by the methods of the "Security setup" interface class; see IEC 62056-6-2:2021, 4.4.7.

The parties generate an ephemeral key pair from the domain parameters D. The parties exchange ephemeral public keys and then compute the shared secret Z using the domain parameters, their ephemeral private key and the ephemeral public key of the other party. The secret keying material is derived using the key derivation function specified in 5.3.4.6.5 from the shared secret Z and other input.

The process is specified in detail in NIST SP 800-56A Rev. 2: 2013, 6.1.2.2 and NSA2, 3.1 and it is shown in Figure 20 and Table 4 below.



NOTE This figure reproduces NSA2, Figure 1.

Figure 20 – C(2e, 0s) scheme: each party contributes only an ephemeral key pair

Prerequisites:

1) each party has an authentic copy of the same set of domain parameters, D. D shall be selected from one of the two sets of domain parameters, see Annex G;

2) the parties have agreed on using the NIST Concatenation KDF; see 5.3.4.6.5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384;

3) prior to or during the key agreement process, the parties obtain the identifier associated with the other party during the key agreement scheme.

NOTE See also NIST SP 800-56A Rev. 2: 2013 and NSA2 for additional information on assurance on the validity of the domain parameters, the validity of the private and public key and the assurance of the possession of the private key.

Table 4 – Ephemeral Unified Model key agreement scheme summary

	Party U	**Party V**
Domain parameters	$(q, FR, a, b\{,SEED\}, G, n, h)$ as determined by the security suite	$(q, FR, a, b\{,SEED\}, G, n, h)$ as determined by the security suite
Static data	N/A	N/A
Ephemeral data	Ephemeral private key, $d_{e,U}$ Ephemeral public key, $Q_{e,U}$	Ephemeral private key, $d_{e,V}$ Ephemeral public key, $Q_{e,V}$
Computation	Compute Z by calling ECC CDH using $d_{e,U}$ and $Q_{e,V}$	Compute Z by calling ECC CDH using $d_{e,V}$ and $Q_{e,U}$
Derive secret keying material	1. Compute kdf(Z, $OtherInput$) 2. Destroy Z	1. Compute kdf(Z, $OtherInput$) 2. Destroy Z
NOTE This table is based on NIST SP 800-56A Rev. 2: 2013 Table 18 and NSA2 Table 1.		

The rationale for choosing a C(2e, 0s) scheme is specified in NIST SP 800-56A Rev. 2: 2013, 8.2.

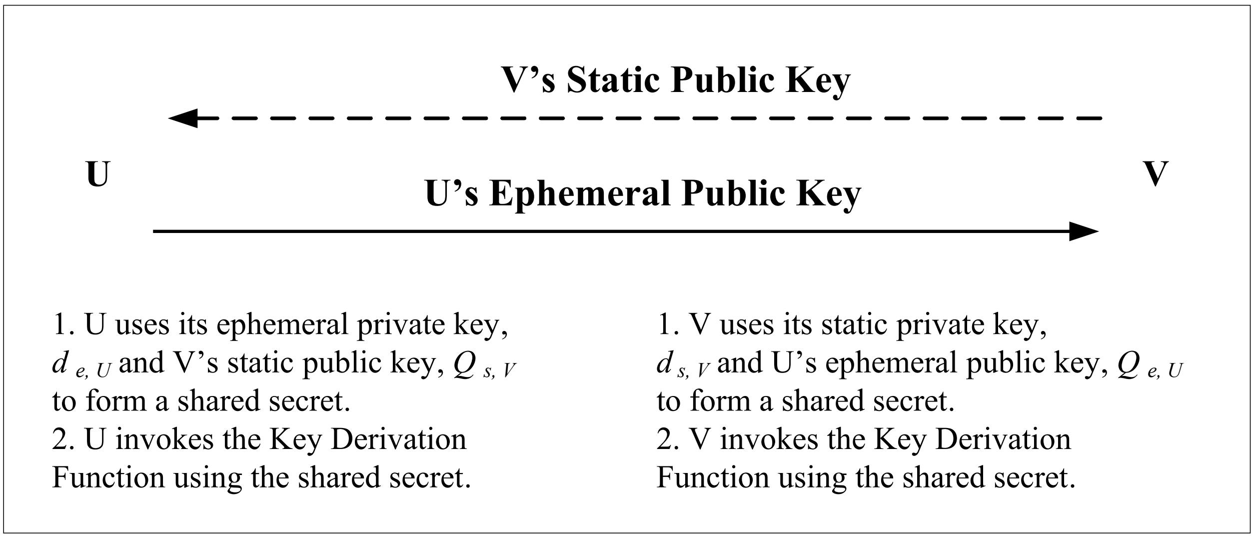
The use of this scheme in DLMS®/COSEM is further explained in I.1.

5.3.4.6.3 The One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme

This scheme is for use by a DLMS®/COSEM server and another party to agree on an ephemeral encryption key to protect xDLMS APDUs or COSEM data. The party sending the message (the originator) plays the role of party U and the other party (the recipient) plays the role of party V.

NOTE 1 The terms originator and recipient refer to fields of the general-ciphering APDU.

For this scheme, party U generates an ephemeral key pair; party V has only a static key pair. Party U obtains Party V's static public key in a trusted manner (for example, from a certificate signed by a trusted CA) and sends its ephemeral public key to party V. Each party derives the shared secret Z by using its own private key and the other party's public key. Each party derives secret keying material using the key derivation method specified in 9.2.3.4.6.5 from the shared secret Z and other input.



NOTE This figure reproduces NSA2, Figure 2.

Figure 21 – C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair

The process is specified in detail in NIST SP 800-56A Rev. 2: 2013, 6.2.2.2 and NSA2, 3.2 and is shown in Figure 21 and Table 5.

Prerequisites:

1) each party shall have an authentic copy of the same set of domain parameters, D. D shall be selected from one of the two sets of domain parameters, see Annex G;

4) party V shall have been designated as the owner of a static key pair that was generated as specified in 5.6.2 using the set of domain parameters, D;

5) the parties have agreed on using the NIST Concatenation KDF, see 5.3.4.6.5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384;

6) prior to or during the key agreement process, the parties obtain the identifier associated with the other party during the key agreement scheme. Party U shall obtain the static public

key that is bound to party V's identifier. This static public key shall be obtained in a trusted manner (e.g., from a certificate signed by a trusted CA).

NOTE 2 See also NIST SP 800-56A Rev. 2: 2013 and NSA2 for additional information on assurance on the validity of the domain parameters, the validity of the private and public key and the assurance of the possession of the private key.

Table 5 – One-pass Diffie-Hellman key agreement scheme summary

	Party U	**Party V**
Domain Parameters	$(q, FR, a, b\{,SEED\}, G, n, h)$ As determined by the security suite	$(q, FR, a, b\{,SEED\}, G, n, h)$ As determined by the security suite
Static Data	N/A	Static private key, $d_{s,\ V}$ Static public key, $Q_{s,\ V}$
Ephemeral Data	Ephemeral private key, $d_{e,\ U}$ Ephemeral public key, $Q_{e,\ U}$	N/A
Computation	Compute Z by calling ECC CDH using $d_{e,\ U}$ and $Q_{s,\ V}$	Compute Z by calling ECC CDH using $d_{s,\ V}$ and $Q_{e\ ,U}$
Derive Secret Keying Material	1. Compute kdf(Z, $OtherInput$) 2. Destroy Z	1. Compute kdf(Z, $OtherInput$) 2. Destroy Z
NOTE This table is based on NIST SP 800-56A Rev. 2: 2013 Table 24 and **NSA2** Table 2.		

The rationale for choosing this scheme is specified in NIST SP 800-56A Rev. 2: 2013, 8.4.

The use of this scheme in DLMS®/COSEM is further explained in Clause I.2.

5.3.4.6.4 The Static Unified Model C(0e, 2s, ECC CDH) scheme

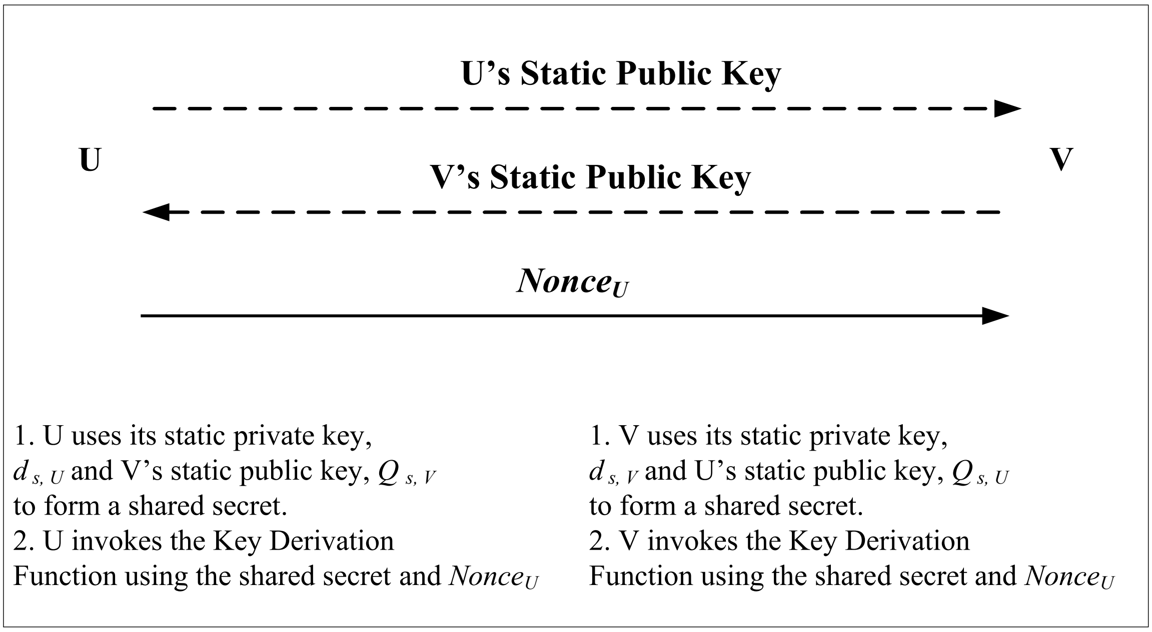
This scheme is for use by a DLMS®/COSEM server and another party to agree on an ephemeral encryption key to protect xDLMS APDUs or COSEM data. The party sending the message (the originator) plays the role of party U and the other party (the recipient) plays the role of party V.

NOTE 1 The terms originator and recipient refer to fields of the general-ciphering APDU.

In this case, the parties use only static key pairs. Each party obtains the other party's static public key. A nonce, $Nonce_U$, is sent by party U to party V to ensure that the derived keying material is different for each key-establishment transaction.

The parties derive the shared secret Z using their own static private key and the other party's static public key. Secret keying material is derived using the key-derivation function specified in 5.3.4.6.5 the shared secret Z, U and V's identifier and the nonce.

The process is shown in Figure 22 and Table 6.



NOTE This figure is based on NIST SP 800-56A Rev. 2: 2013, Figure 15.

Figure 22 – C(0e, 2s) scheme: each party contributes only a static key pair

Prerequisites:

1) each party shall have an authentic copy of the same set of domain parameters, D. D must be selected from one of the two sets of domain parameters, see Annex G;

7) each party has been designated as the owner of a static key pair that was generated as specified in 5.6.2 using the set of domain parameters, D;

8) the parties have agreed on using the NIST Concatenation KDF, see 5.3.4.6.5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384;

9) prior to or during the key agreement process, each party shall obtain the identifier associated with the other party during the key agreement scheme;

10) both parties shall obtain the static public key of the other party that is bound to the identifier. These static public keys shall be obtained in a trusted manner (e.g., from a certificate signed by a trusted CA.

NOTE 2 See also NIST SP 800-56A Rev. 2: 2013 for additional information on assurance on the validity of the domain parameters, the validity of the private and public key and the assurance of the possession of the private key.

Table 6 – Static Unified Model key agreement scheme summary

	Party U	**Party V**
Domain Parameters	*(q, FR, a, b{,SEED}, G, n, h)* As determined by the security suite	*(q, FR, a, b{,SEED}, G, n, h)* As determined by the security suite
Static Data	Static private key, $d_{s,\ U}$ Static public key, $Q_{s,\ U}$	Static private key, $d_{s,\ V}$ Static public key, $Q_{s,\ V}$
Ephemeral Data	$Nonce_U$	
Computation	Compute Z by calling ECC CDH using $d_{s,\ U}$ and $Q_{s,\ V}$	Compute Z by calling ECC CDH using $d_{s,\ V}$ and $Q_{s\ ,U}$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* using $Nonce_U$ 2. Destroy Z	1. Compute *DerivedKeyingMaterial* using $Nonce_U$ 2. Destroy Z

NOTE 1 This table is based on NIST SP 800-56A Rev. 2: 2013, Table 26.

NOTE 2 The value of Z is the same in all C(0e, 2s) key-establishment transactions between the same two parties, therefore if it is ever compromised, then all of the keying material derived in past, current, and future C(0e, 2s) key-agreement transactions between these same two entities that employ these same static key pairs may be compromised as well. Any shared secret Z that is not 'zeroized' shall be stored and used with the same security protections as private keys.

The rationale for choosing this scheme is specified in NIST SP 800-56A Rev. 2: 2013, 8.5.

The use of this scheme in DLMS®/COSEM is further explained in Clause I.3.

5.3.4.6.5 Key Derivation Function – The NIST Concatenation KDF

The NIST Concatenation KDF as specified in NIST SP 800-56A Rev. 2: 2013, 5.8.1.1 and NSA2, Clause 5 shall be used. It is as follows:

Function call: kdf(*Z, OtherInput*)

where *OtherInput* consists of *keydatalen* and *OtherInfo*.

Implementation-Dependent Parameters:

1) *hashlen*: an integer that indicates the length (in bits) of the output of the hash function, *hash*, used to derive blocks of secret keying material. This will be 256 (for SHA-256) in the case of security suite 1 and 384 (for SHA-384) in the case of security suite 2;
2) *max_hash_inputlen*: an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

The length shall be less than 264 bits for SHA-256 and less than 2128 bits for SHA-384.

Function: H: a hash function: SHA-256 in the case of security suite 1 and SHA-384 in the case of security suite 2.

Input:

1) Z: a byte string that represents the shared secret z;
2) *keydatalen*: an integer that indicates the length (in bits) of the secret keying material to be generated: 128 bit for security suite 1 and 256 bit for security suite 2;
3) *OtherInfo:* A bit string equal to the following concatenation:

AlgorithmID || *PartyUInfo* || *PartyVInfo* {||*SuppPubInfo*}{||*SuppPrivInfo*}

where the subfields are defined as follows:

1) *AlgorithmID*: A bit string that indicates how the derived secret keying material will be parsed and for which algorithm(s) the derived secret keying material will be used. See Table 8;

2) *PartyUInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by Party U to the key derivation process;

3) *PartyVInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by Party V to the key derivation process;

11) (Optional) *SuppPubInfo*: A bit string containing additional, mutually known public information. Not used in DLMS®/COSEM;

12) (Optional) *SuppPrivInfo*: A bit string containing additional, mutually known private information (for example, a shared secret symmetric key that has been communicated through a separate channel). Not used in DLMS®/COSEM.

The format and content of each subfield and substring is specified in Table 7.

Table 7 – *OtherInfo* subfields and substrings

Subfield *Substring*	Key agreement scheme			Length in octets	Value
	C(2e, 0s)	C(1e, 1s)	C(0e, 2s)		
	Format				
AlgorithmID	Fixed	Fixed	Fixed	7	See Table 8
PartyUinfo	Fixed	Fixed	Variable	8+n	–
ID_U	Fixed	Fixed	Fixed	8	originator-system-title
NonceU	–	–	Variable	n	*Datalen* = length of transaction-id, shall be 1 octet *Data* = value of transaction-id
PartyVInfo	Fixed	Fixed	Fixed	8	–
ID_V	Fixed	Fixed	Fixed	8	recipient-system-title
"originator-system-title", transaction-id" and "recipient-system-title" are the fields of the general-ciphering APDU					

In DLMS®/COSEM, key derivation delivers a single key for a given purpose. The length is as determined by the security suite. *AlgorithmId* shall be as specified in Table 8. See also 7.2.2.4.

Table 8 – Security algorithm ID-s

Algorithm	COSEM cryptographic algorithm ID	Encoded value
AES-GCM-128	2.16.756.5.8.3.0	60 85 74 05 08 03 00
AES-GCM-256	2.16.756.5.8.3.1	60 85 74 05 08 03 01
AES-WRAP-128	2.16.756.5.8.3.2	60 85 74 05 08 03 02
AES-WRAP-256	2.16.756.5.8.3.3	60 85 74 05 08 03 03

5.3.5 Random number generation

Strong random number generator (RNG) shall be provided to generate the random numbers required for the various algorithms used in DLMS®/COSEM. The RNG shall be preferably non-deterministic. If a non-deterministic RBG is not available, the system shall make use of sufficient entropy to create a good quality seed for a deterministic RNG.

5.3.6 Compression

NOTE Compression does not involve cryptography, but it is a transformation of the xDLMS APDU. For this reason, and as it is controlled together with symmetric key ciphering it is specified here.

Compression can be applied to COSEM data or xDLMS APDUs. This process can be combined with symmetric key ciphering. See 5.7.2.4.7 and 5.7.2.4.8. The compression algorithm shall be as specified in ITU-T V.44: 2000. This algorithm has been selected for to meet the following requirements:

- low processing load;
- low memory requirements;
- low latency.

The use of compression is indicated by bit 7 of the Security Control Byte, specified in 5.7.2.4.

5.3.7 Security suite

A security suite determines the set of cryptographic algorithms available for the various cryptographic primitives and the key sizes.

The DLMS®/COSEM security suites – see Table 9 – are based on NSA Suite B and include cryptographic algorithms for authentication, encryption, key agreement, digital signature and hashing specifically:

- authentication and encryption: the Advanced Encryption Standard (AES) shall be used as specified in FIPS PUB 197, with key sizes of 128 and 256 bits. AES shall be used with the Galois/Counter Mode (GCM) of operation specified in NIST SP 800-38D:2007;
- digital signature: the Elliptic Curve Digital Signature Algorithm (ECDSA) shall be used as specified FIPS PUB 186-4:2013 and in NSA1, using the curves P-256 or P-384;
- key agreement:
  - the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme, see 5.3.4.6.2;
  - the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme, see 5.3.4.6.3; and
  - the Static Unified Model C(0e, 2s, ECC CDH) scheme, see 5.3.4.6.4

  shall be used using the elliptic curves P-256 or P-384.
- hashing: the Secure Hash Algorithms (SHA) SHA-256 and SHA-384 shall be used as specified in FIPS PUB 180-4:2012.

In addition, a key wrapping and a compression algorithm are available.

Table 9 – DLMS®/COSEM security suites

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

5.4 Cryptographic keys – overview

A cryptographic key is a parameter used in conjunction with a cryptographic algorithm that determines its operation in such a way that an entity with knowledge of the key can reproduce or reverse the operation, while an entity without knowledge of the key cannot. In DLMS®/COSEM, examples of operations include:

- the transformation of plaintext into ciphertext;
- the transformation of ciphertext into plaintext;
- the computation and verification of an authentication code (MAC);
- key wrapping;
- applying and verifying digital signature;
- key agreement.

Keys used with symmetric key algorithms are specified in 5.5.

Keys used with public key algorithms are specified in 5.6.

5.5 Key used with symmetric key algorithms

5.5.1 Symmetric keys types

Symmetric keys are classified according to:

b) their purpose:
   1) a key encrypting key (KEK) is used to encrypt / decrypt other symmetric keys; see 5.5.4. In DLMS®/COSEM this is the master key;
   2) an encryption key is used as the block cipher key of the AES-GCM algorithm, see also 5.3.3.7.4;
   3) an authentication key is used as Additional Authenticated Data (AAD) in the AES-GCM algorithm, see also 5.3.3.7.5.

c) their lifetime:
   1) static keys that are intended to be used for a relatively long period of time. In DLMS®/COSEM these may be:
   - a global key that may be used over several AAs established repeatedly between the same partners. A global key may be a unicast encryption key (GUEK), a broadcast encryption key (GBEK) or an authentication key (GAK);
   - a dedicated key that may be used repeatedly during a single AA established between two partners. Therefore, its lifetime is the same as the lifetime of the AA. A dedicated key can be only a unicast encryption key.
   2) ephemeral keys used generally for a single exchange within an AA.

For generation and distribution of symmetric keys, see NIST SP 800-57:2012, 8.1.5.2.

A master key and global keys are established between each DLMS®/COSEM client – server pair using one of the methods shown in Table 10. They should be renewed in appropriate intervals, see 5.3.3.7.3 and 5.5.6.

Dedicated keys are generated by the DLMS®/COSEM client and transported to the server in the dedicated-key field of the xDLMS InitiateRequest APDU, carried by the user-information field of the AARQ APDU. When the dedicated key is present, the xDLMS InitiateRequest APDU shall be authenticated and encrypted using the AES-GCM-128 / 256 algorithm, the global unicast encryption key and – if in use, see 5.3.3.7.5 – the authentication key. The xDLMS InitiateResponse APDU, carried by the user-information field of the AARE APDU shall be also encrypted and authenticated the same way. When the dedicated key is used, the key-set bit of the security control byte, see Table 27 – is not relevant and shall be set to zero.

NOTE The AARQ and the AARE APDUs themselves are not protected.

Table 10 summarizes the symmetric key types, their purpose, the methods to establish them and their use with the different APDUs and between different entities.

Table 10 – Symmetric keys types

Key type	Purpose	Key establishment	Use
Master key, KEK [1]	Key Encrypting Key (KEK) for: - (new) master key; - global encryption or authentication keys; - ephemeral encryption keys.	Out of band	Can be identified as the KEK in general-ciphering APDUs between client-server, see Table 11
		Key wrapping	
		Key agreement [3]	
Global unicast encryption key, GUEK [2]	Block cipher key for unicast - xDLMS APDUs and / or - COSEM Data	Key wrapping	- service-specific global ciphering APDU client-server - general-glo-ciphering APDU client-server - general-ciphering APDU client-server - “Data protection” object protection parameters
		Key agreement [3]	
Global broadcast encryption key, GBEK [2]	Block cipher key for broadcast - xDLMS APDUs and / or - COSEM Data	Key wrapping	
(Global) Authentication key, GAK [2]	Part of AAD to the ciphering process of xDLMS APDUs and / or COSEM data	Key wrapping	All APDUs between client-server and third party-server
		Key agreement [3]	
Dedicated key (unicast)	Block cipher key of unicast xDLMS APDUs, within an established AA	Key-transport in xDLMS Initiate.request APDU	- service-specific dedicated ciphering APDU client-server - general-ded-ciphering APDU client-server during the lifetime of an AA
Ephemeral encryption key	Block cipher key for: - xDLMS APDUs and/or - COSEM data	Key wrapping	- general-ciphering APDU client-server - “Data protection” object protection parameters
	Block cipher key for: - xDLMS APDUs and/or - COSEM data	Key agreement [4]	- general-ciphering APDU client-server or third-party server - “Data protection” object protection parameters

1) Held by a “Security setup” object. Different AAs may use the same or different “Security setup” objects.

2) Held by a “Security setup” object. Different AAs may use the same or different “Security setup” objects. The use of the GUEK or the GBEK can be identified by:
   – the key-set bit of the Security Control Byte, see Table 27; or
   – by the key-id parameter of the general-ciphering APDU, see Table 11;
   – or by the protection parameters of the “Data protection “ IC, see IEC 62056-6-2:2021, 4.4.9.

3) Established using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme, see 5.3.4.6.2

4) Established using the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme or the Static Unified Model C(0e, 2s, ECC CDH) scheme. See 5.3.4.6.3 and 5.3.4.6.4.

5.5.2 Key information with general-ciphering APDU and data protection

When the general-ciphering APDU is used to protect xDLMS APDUs or when COSEM data is protected, the sender sends the necessary information on the key that has been / shall be used to cipher / decipher the xDLMS APDU / COSEM data, together with the ciphered xDLMS APDU / COSEM data.

The key information required is summarized in Table 11 and further specified in 5.5.3, 5.5.4 and 5.5.5.

Table 11 – Key information with general-ciphering APDU and data protection

Key information choices		Comment
key-Info		
identified-key	S	The EK is identified
key-id	M	
global-unicast-encryption-key	S	GUEK
global-broadcast-encryption-key	S	GBEK
wrapped-key	S	The EK is transported using key wrapping
kek-id	M	
master-key	M	Identifies the key used for wrapping the key-ciphered-data. 0 = Master Key (KEK)
key-ciphered-data	M	Randomly generated key wrapped with KEK
agreed-key	S	The key is agreed by the parties using either: - the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme see 5.3.4.6.3; or - the Static Unified Model C(0e, 2s, ECC CDH) scheme see 5.3.4.6.4.
key-parameters	M	Identifier of the key agreement scheme: 0x01: C(1e, 1s ECC CDH) 0x02: C(0e, 2s ECC CDH) All other reserved.
key-ciphered-data	M	- In the case of the C(1e, 1s, ECC CDH) scheme: the public key $Q_{e,U}$, of the ephemeral key agreement key pair of party U, signed with the private digital signature key of party U. - In the case of the C(0e, 2s, ECC CDH) scheme: an octet-string of length zero. In this case party U has to provide a nonce, $Nonce_U$. See 5.3.4.6.4 and 5.3.4.6.5.
M: Mandatory (part of a SEQUENCE) S: Selectable (part of a CHOICE) For the ASN.1 specification, see Clause 8.		
NOTE Using key identification restricts exchanging protected xDLMS APDUs / COSEM data between a client and a server because the GUEK and the GBEK shall not be disclosed to any party other than the client and the server.		

5.5.3 Key identification

The key identified may be the Global Unicast Encryption Key (GUEK) or the Global Broadcast Encryption Key (GBEK). In this case, the key-set bit of the security control byte – see Table 27 – is not relevant and shall be set to zero.

5.5.4 Key wrapping

Key wrapping can be used to establish static or ephemeral symmetric keys.

The algorithm is the AES key wrap algorithm specified in 5.3.3.8. The KEK is the master key. Consequently, this method can be used only between parties sharing the master key, i.e. between a client and a server.

The static keys that can be established using key wrapping may be:

- the master key, KEK; and/or
- the global unicast encryption key GUEK; and/or
- the global broadcast encryption key GBEK; and/or
- the (global) authentication key, GAK.

To establish these static keys using key wrap, the key shall be first generated by the client, then it shall be transferred to the server by invoking the *key_transfer* method of the "Security setup" object, see IIEC 62056-6-2:2021, 4.4.7. The method invocation parameter shall carry the key_id(s) and the wrapped key(s). The APDU carrying the service that invokes the method and the method invocation parameters shall be protected as required by the security policy and the access rights.

NOTE The required level of protection can be specified in project specific companion specifications.

To establish an ephemeral key using key wrapping, the originator of the xDLMS APDU or the COSEM data randomly generates an ephemeral key. This key shall be wrapped using the AES key wrap algorithm and the KEK and shall be sent to the recipient together with the xDLMS APDU or the COSEM data that have been ciphered using the ephemeral key. The recipient shall unwrap the key then it shall use it to decipher the xDLMS APDU / COSEM data received.

5.5.5 Key agreement

Key agreement can be used to establish static keys between a server and a client or ephemeral keys between a server and a client or a third party. Different key agreement schemes are available to establish different keys.

The Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme can be used by the client and the server to agree on the:

- master key, KEK; and/or
- global unicast encryption key GUEK; and/or
- global broadcast encryption key GBEK; and/or
- (global) authentication key, GAK.

This scheme is supported by the *key_agreement* method of the "Security setup" interface class, see IEC 62056-6-2:2021, 4.4.7. The method invocation parameters carry the necessary parameters as specified in 5.3.4.6.2. The APDUs carrying the service that invokes the method as well as the method invocation parameters shall be protected as required by the security policy and the access rights. See also Annex I.

NOTE 1 The required level of protection can be specified in project specific companion specifications.

To establish an ephemeral encryption key – used as the block cipher key – using key agreement, two schemes are available:

- the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme specified in 5.3.4.6.3;

  NOTE 2 Unless specified otherwise in a project specific companion specification, the C(1e, 1s ECC CDH) scheme shall be used.

- the Static Unified Model C(0e, 2s, ECC CDH) scheme specified in 5.3.4.6.4.

5.5.6 Symmetric key cryptoperiods

Symmetric key cryptoperiods should be determined in project specific companion specifications. Recommendations are given in NIST SP 800-57:2012, Part 1, 5.3.5 *Symmetric Key Usage Periods and Cryptoperiods* and 5.3.6 *Cryptoperiod Recommendations for Specific Key Types.*

5.6 Keys used with public key algorithms

5.6.1 Overview

Asymmetric keys – see Table 12 – are classified according to:

- their purpose: digital signature key or key agreement key;
- by their lifetime: static keys or ephemeral keys.

Table 12 – Asymmetric keys types and their use

Digital signature			
Key type	**Signatory**	**Verifier**	**Use**
Digital signature key pair	Private key	Public key	Signatory uses private key to compute digital signature: - on an xDLMS APDUs; and/ or - on COSEM data; or - on an ephemeral public key agreement key. Verifier uses public key to verify digital signature - on an xDLMS APDUs; and/or - on COSEM data; or - on an ephemeral public key agreement key received.
Key agreement			
Key type	**Party U**	**Party V**	**Use**
Ephemeral key agreement key pair	Private key Public key	Private key Public key	In the case of the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme both parties have an ephemeral key pair. See 5.3.4.6.2. In the case of the One-Pass Diffie-Hellman C(1e,1s, ECC CDH) scheme only party U has an ephemeral key pair. See 5.3.4.6.3
Static key agreement key pair	Private key Public key	Private key Public key	In the case of the One-Pass Diffie-Hellman C(1e,1s, ECC CDH) scheme only party V has a static key pair. See 5.3.4.6.3. In the case of the Static Unified Model, C(0e, 2s, ECC CDH) scheme both the party U and party V have a static key pair. See 5.3.4.6.4.

5.6.2 Key pair generation

An ECC key pair d and Q is generated for a set of domain parameters *(q, FR, a, b {, domain_parameter_seed}, G, n, h)*. Two methods are provided for the generation of the ECC private key d and public key Q; one of these two methods shall be used to generate d and Q.

Prior to generating ECDSA key pairs, assurance of the validity of the domain parameters *(q, FR, a, b {, domain_parameter_seed}, G, n, h)* shall have been obtained.

For details, see FIPS PUB 186-4:2013, Annex B.4.

5.6.3 Public key certificates and infrastructure

5.6.3.1 Overview

This subclause 5.6.3 describes the public key certificates for the purposes of DLMS®/COSEM and an example PKI infrastructure to manage them. It is based on the following documents:

- NIST SP 800-21:2005 and NIST SP 800-32:2001, providing a general description of public key cryptography and public key infrastructures;
- ITU-T X.509:2008, specifying public key and attribute certificate frameworks;
- RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile;
- NSA3 specifying the NSA Suite B Base Certificate and CRL Profile.

The trust model is described in 5.6.3.2.

A PKI architecture – as an example – is described in 5.6.3.3.

The certificate and certificate extension profile is specified in 5.6.4.

The public key certificates to be held by DLMS®/COSEM servers are specified in 5.6.5.

The management of certificates is specified in 5.6.6.

5.6.3.2 Trust model

For DLMS®/COSEM based meter data exchange systems using public key cryptography various public-private key pairs and public key certificates should be in place.

A public key certificate binds a public key to an identity: the subject. A certificate is digitally signed by a Certification Authority.

To provide and manage the certificates, some form of Public Key Infrastructure is required. A PKI consists of Certification Authorities issuing certificates and end entities using these certificates. For a PKI example, see 5.6.3.3.

In its simplest form, a certification hierarchy consists of a single CA. However, the hierarchy usually contains multiple CAs that have clearly defined parent-child relationships. It is also possible to deploy multiple hierarchies.

The PKI needs a trust anchor that is used to validate the first certificate in a sequence of certificates. The trust anchor may be a Root-CA certificate, a Sub-CA certificate or a directly trusted key.

NOTE 1 Trust anchors are Certificates or directly trusted keys marked as such. However, this marking is out of the Scope of this International Standard.

DLMS®/COSEM servers shall be provisioned with one or more trust anchors during manufacturing using a trusted Out of Band (OOB) process.

NOTE 2 Provisioning clients and third parties with trust anchors is out of the Scope of this International Standard.

DLMS®/COSEM servers may also be provisioned with their own certificates and certificates of CAs, DLMS®/COSEM clients and third parties. This may also happen using a trusted OOB process or through the “Security setup” object.

The “Security setup” interface class – see IEC 62056-6-2:2021, 4.4.7 – provides:

- an attribute that provides information on the certificates stored on the server;
- a method to generate server key pairs and a method to generate Certificate Signing Request (CSR) information on the server to be sent by the client to a CA;
- methods to import, export and remove certificates.

These attributes / methods can be used to manage the certificates by the client or by third parties via the client acting as a broker. Messages accessing the attributes and methods of "Security setup" objects and the data included shall be suitably protected.

Certificates generally have a validity period. However, certificates issued to DLMS®/COSEM servers may be indefinitely valid. Certificates may be replaced when they expire.

Before a server uses a Certificate, it has to be verified. Verification includes:

- checking syntactic validity of the certificate;
- checking the attributes included in the certificate;
- checking that the certificate validity period has not expired;
- checking the certification path to the trust anchor;
- checking the signature of the issuer of the certificate.

It is assumed that the trust anchor, other CA-certificates, as well as the certificates of DLMS®/COSEM clients and third parties held by the server are all valid. It is the responsibility of the system to replace / remove any certificates the validity of which have expired or that have been revoked.

Clients and third parties also have to verify server certificates before using them. They may have capabilities to verify the status of certificates they are using. However this is outside the Scope of this International Standard.

5.6.3.3 PKI architecture – informative

5.6.3.3.1 General

NOTE 1 The introductory text is quoted from NIST SP 800-21:2005, 3.7.

A PKI is a security infrastructure that creates and manages public key certificates to facilitate the use of public key (i.e., asymmetric key) cryptography. To achieve this goal, a PKI needs to perform two basic tasks:

1) Generate and distribute public key certificates to bind public keys to other information after validating the accuracy of the binding; and
2) Maintain and distribute certificate status information for unexpired certificates.

For DLMS®/COSEM a hierarchical PKI is foreseen, comprising the following components as shown in Figure 23.

NOTE 2 The actual structure of the PKI is left to project specific companion specifications to meet the operators' needs.

- Root Certification Authority (Root-CA): see 5.6.3.3.2;
- Certification Authority / Subordinate Authority (Sub-CA): see 5.6.3.3.3;
- End entities: entities that do not issue certificates: DLMS®/COSEM clients, DLMS®/COSEM servers and third parties: see 5.6.3.3.4.

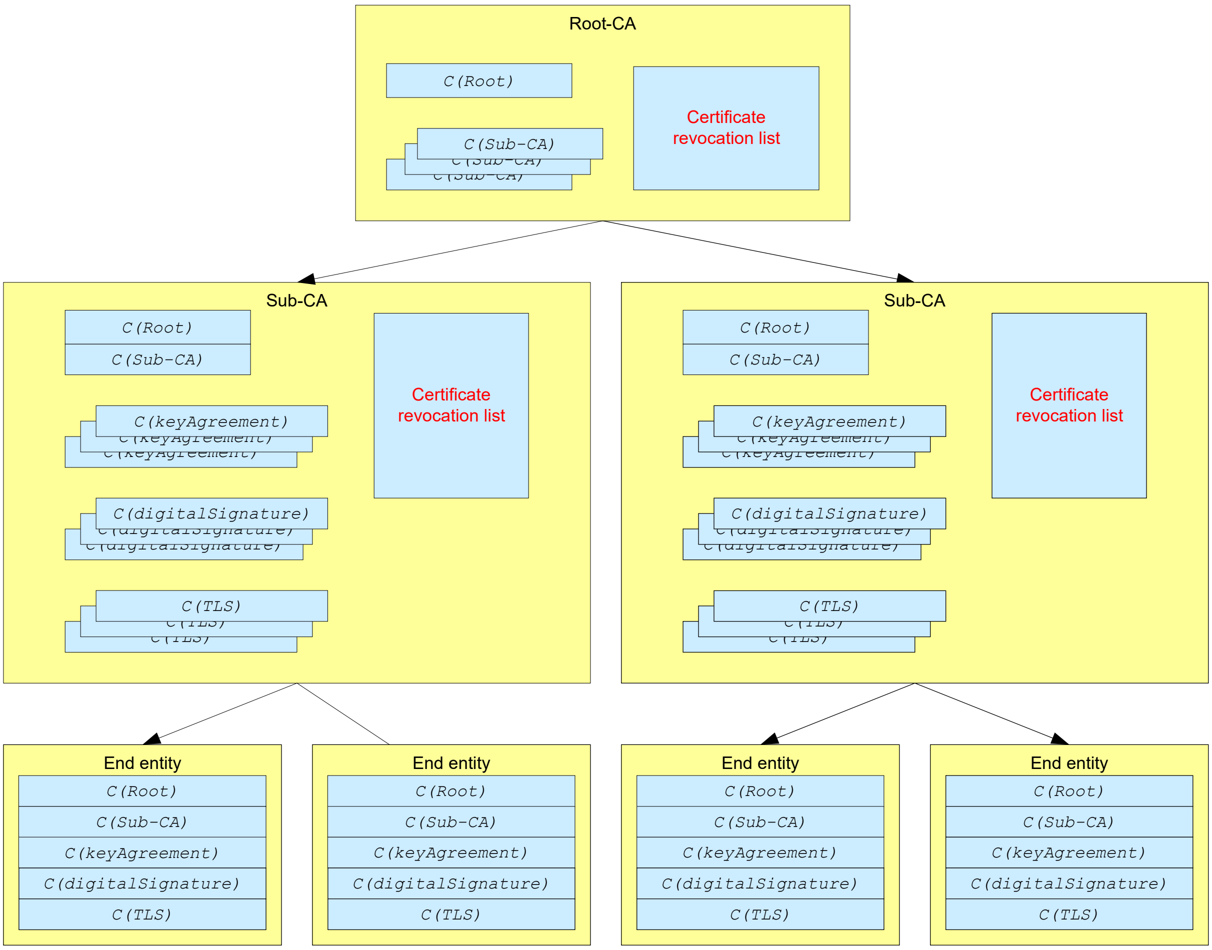


Figure 23 – Architecture of a Public Key Infrastructure (example)

5.6.3.3.2 Root-CA

The Root-CA provides the trust anchor of the PKI. It issues certificates for Sub-CAs and maintains a certificate revocation list (CRL). The Root-CA Certificate Policy defines the rules for handling the issuance of certificates.

The Root-CA owns the Root certificate `C(Root)`. The Certificate of the Root-CA is self-signed with the Root-CA private key. Sub-CA certificates are also signed with the Root-CA private key.

5.6.3.3.3 Sub-CA

A Sub-CA is an organization that issues certificates for end entities. Each Sub-CA is authorised by the Root-CA to do so.

NOTE Sub-CAs may be independent organizations, or may be meter market participants, meter operators, manufacturers.

Each Sub-CA must handle a Certificate Policy, which must comply with the Root-CA Certificate Policy.

Sub-CAs also maintain the list of certificates issued to end entities and a Certification Revocation List.

A sub-CA owns a certificate *C(Sub-CA)*. This certificate is issued by the Root-CA. The private key of the Sub-CA is used for signing end entity certificates.

5.6.3.3.4 End entities

In the PKI infrastructure, an end entity is a user of PKI certificates and/or an end user system that is the subject of a certificate. The following end entity certificates are defined:

- digital signature key certificate `C(digitalSignature)`, used for digital signature;
- static key agreement key certificate `C(keyAgreement)`, used for key agreement;
- [optional] TLS-Certificate `C(TLS)`, used for performing authentication between a DLMS®/COSEM client and a DLMS®/COSEM server prior the establishment of a TLS secure channel.

See also 5.6.5.

5.6.4 Certificate and certificate extension profile

5.6.4.1 General

This subclause 5.6.4 specifies the certificate and certification extension profile as required for DLMS®/COSEM systems using public key cryptography.

All certificates shall have the structure specified for X.509 version 3 certificates.

For the tables presenting the fields of the certificate and certificate extensions, the following notation applies:

- m (mandatory): the field must be filled;
- o (optional): the field is optional;
- x (do not use): the field shall not be used.

Each certificate extension is designated either as critical or non-critical. A certificate-using system MUST reject the certificate if it encounters a critical extension it does not recognize or a critical extension that contains information that it cannot process. A non-critical extension may be ignored if it is not recognized, but MUST processed if it is recognized.

This International Standard specifies minimum requirements. Project specific companion specifications may specify more strict requirements, for example a field specified in this International Standard as optional may be made mandatory or an extension designated as non-critical may be designated as critical.

5.6.4.2 The X.509 v3 Certificate

An X.509 v3 certificate is a SEQUENCE of three required fields as shown in Table 13.

Table 13 – X.509 v3 Certificate structure

Certificate field	RFC 5280 reference	m/x/o	Value
`Certificate`	4.1.1		
`tbsCertificate`	4.1.1.1	m	See below
`signatureAlgorithm`	4.1.1.2	m	See below
`signatureValue`	4.1.1.3	m	See below

The `tbsCertificate` field contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information. The fields of the `tbsCertificate` are summarized in Table 14. The `tbsCertificate` usually includes extensions; these are described in 5.6.4.4.

The `signatureAlgorithm` field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate.

```
AlgorithmIdentifier::= SEQUENCE {
     algorithm               OBJECT IDENTIFIER
     parameters              ANY DEFINED BY algorithm OPTIONAL }
```

The two algorithm identifiers used in DLMS®/COSEM are:

- `ecdsa-with-SHA256`, OID 1.2.840.10045.4.3.2 in the case of security suite 1;
- `ecdsa-with-SHA384`, OID 1.2.840.10045.4.3.3 in the case of security suite 2;

The `signatureValue` contains a digital signature computed upon the ASN.1 DER encoded `tbsCertificate`. The ASN.1 DER encoded tbsCertificate is used as the input to the signature function.

By generating this signature, a CA certifies the validity of the information in the `tbsCertificate` field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

5.6.4.3 tbsCertificate

5.6.4.3.1 Overview

The fields of the `tbsCertificate` are shown in Table 14.

Table 14 – X.509 v3 tbsCertificate fields

Certificate field	RFC 5280 reference	Clause	m/x/o	Comment
`tbsCertificate`	4.1.2	5.6.4.2		
`Version`	4.1.2.1	–	m	‘v3’ (value is 2)
`Serial Number`	4.1.2.2	5.6.4.3.2	m	Certificate serial number assigned by the CA (not longer than 20 octets)
`Signature`	4.1.2.3	–	m	Same algorithm identifier as the `signatureAlgorithm` in the Certificate
`Issuer`	4.1.2.4	5.6.4.3.3	m	Distinguished name (DN) of the certificate issuer.
`Validity`	4.1.2.5	5.6.4.3.4	m	Validity of the certificate.
`Subject`	4.1.2.6	5.6.4.3.3	m	Distinguished name (DN) of the certificate subject.
`Subject Public Key Info`	4.1.2.7	5.6.4.3.5	m	
`Issuer Unique ID`	4.1.2.8		x	Not applicable
`Subject Unique ID`	4.1.2.8	5.6.4.3.6	o	
`Extensions`	4.1.2.9	5.6.4.4	m	

5.6.4.3.2 Serial number

As specified in RFC 5280, 4.1.2.2 the serial number MUST be a positive integer assigned by the CA to each certificate. It MUST be unique for each certificate issued by a given CA (i.e., the issuer name and serial number identify a unique certificate).

Certificate users MUST be able to handle `serialNumber` values up to 20 octets. Conforming CAs MUST NOT use `serialNumber` values longer than 20 octets.

5.6.4.3.3 Issuer and Subject

The `Issuer` field identifies the entity that has signed and issued the certificate.

The `Subject` field identifies the entity associated with the public key stored in the subject public key field. The subject name MAY be carried in the `Subject` field and/or the `subjectAltName` extension. If subject naming information is present only in the `subjectAltName` extension then the subject name MUST be an empty sequence and the `subjectAltName` extension MUST be critical. See 5.6.4.4.6.

The naming scheme of the various entities of the PKI is shown in the following tables. The names shall be inserted in the `Issuer` or the `Subject` field of the `tbsCertificate` as applicable. See Table 15, Table 16 and Table 17.

Table 15 – Naming scheme for the Root-CA instance (informative)

Attribute	Abbreviation	m/x/o	Name	Comment
`Common Name`	CN	m	<Root-CA>	Name of Root-CA
`Organization`	O	o	<PKI-Name>	Name of PKI
`Organizational Unit`	OU	o		Name of organizational unit
`Country`	C	o		ISO 3166 country code
NOTE Values within the less-than – greater-than signs "< >" are to be assigned by the PKI or the CA as applicable.				

Table 16 – Naming scheme for the Sub-CA instance (informative)

Attribute	Abbrev.	m/x/o	Name	Comment
`Common Name`	CN	m	<XXX-CA>	Name of sub-CA The CN shall finish with "CA" so that the CA function is recognized.
`Organization`	O	o	<PKI-Name>	Name of PKI
`Organizational Unit`	OU	o		Name of organizational unit
`Country`	C	o		ISO 3166 country code
`Locality`	L	o	<Locality>	Locality where the Sub-CA is located
`State`	ST	o	<State>	
NOTE Values within the less-than – greater-than signs "< >" are to be assigned by the PKI or the CA as applicable.				

The format of the elements of the naming scheme of Root-CA and Sub-CA instances is left to project specific companion specifications.

Table 17 – Naming scheme for the end entity instance

Attribute	Abbrev.	m/x/o	Name	Comment
Common Name	CN	m	<System-title>	DLMS®/COSEM System title: 8 bytes represented as a 16 characters: Example: “4D4D4D0000BC614E”
Organization	O	o	<PKI-Name>	Name of PKI
Organizational Unit	OU	o		Name of organizational unit
Country	C	o		ISO 3166 country code
Locality	L	x	<Locality>	Locality where the entity is located
State	S	x	<State>	
NOTE Values within the less-than – greater-than signs “< >” are to be assigned by the PKI or the CA as applicable.				

5.6.4.3.4 Validity period

The certificate validity period is the time interval during which the CA warrants that it will maintain information about the status of the certificate. The field is represented as a SEQUENCE of two dates:

- the date on which the certificate validity period begins (`notBefore`); and
- the date on which the certificate validity period ends (`notAfter`).

In the case of CA certificates, Sub-CA certificates and end-entities other than DLMS®/COSEM servers `notBefore` and `notAfter` shall be well defined dates.

DLMS®/COSEM servers may be given certificates for which no good expiration date can be assigned; such a certificate is intended to be used for the entire lifetime of the device.

To indicate that a certificate has no well-defined expiration date, the `notAfter` should be assigned the GeneralizedTime value of 99991231235959Z.

For details, see RFC 5280, 4.1.2.5.

5.6.4.3.5 SubjectPublicKeyInfo

The field `SubjectPublicKeyInfo` shall have the following structure:

```
SubjectPublicKeyInfo::= SEQUENCE {
     Algorithm                        AlgorithmIdentifier,
     subjectPublicKey                 BIT STRING }
```

An algorithm identifier is defined by the following ASN.1 structure:

```
AlgorithmIdentifier::= SEQUENCE {
     algorithm                        OBJECT IDENTIFIER,
     parameters                       ANY DEFINED BY algorithm OPTIONAL }
```

The algorithm identifier is used to identify a cryptographic algorithm. The OBJECT IDENTIFIER component identifies the algorithm (such as DSA with SHA-1). The contents of the optional parameters field will vary according to the algorithm identified. This field MUST contain the same algorithm identifier as the signature field in the sequence `tbsCertificate`; see 5.6.4.2.

The algorithm `OBJECT IDENTIFIER` shall contain the value:

- OID value: 1.2.840.10045.2.1;
- OID description: ECDSA and ECDH Public Key.

The value of the `parameter` shall be 1.2.840.10045.3.1.7 for the curve NIST P-256 and 1.3.132.0.34 for the curve NIST P-384.

The subjectPublicKey from SubjectPublicKeyInfo is the ECC public key. ECC public keys have the following syntax:

ECPoint::= OCTET STRING

Implementations of Elliptic Curve Cryptography according to this International Standard MUST support the uncompressed form and MAY support the compressed form of the ECC public key. The hybrid form of the ECC public key from [X9.62] must not be used.

As specified in SEC1:2009:

- The elliptic curve public key (a value of type ECPoint that is an OCTET STRING) is mapped to a subjectPublicKey (a value of type BIT STRING) as follows: the most significant bit of the OCTET STRING value becomes the most significant bit of the BIT STRING value, and so on; the least significant bit of the OCTET STRING becomes the least significant bit of the BIT STRING. Conversion routines are found in SEC1:2009, 2.3.1 and 2.3.2;
- The first octet of the OCTET STRING indicates whether the key is compressed or uncompressed. The uncompressed form is indicated by 0x04 and the compressed form is indicated by either 0x02 or 0x03 (see 2.3.3 in SEC1:2009. The public key MUST be rejected if any other value is included in the first octet.

5.6.4.3.6 Subject Unique ID

Subject unique IDs may be optionally used in end device certificates other than server certificates. The use of this field is left to project specific companion specifications.

5.6.4.4 Certificate extensions

5.6.4.4.1 Overview

The extensions defined for X.509 v3 certificates provide methods for associating additional attributes with users or public keys and for managing relationships between CAs. Each extension in a certificate is designated as either critical (TRUE) or non-critical (FALSE).

The extension fields have to be completed according to the type of Certificate used, as specified in Table 18.

Table 18 – X.509 v3 Certificate extensions

	Attributes	RFC 5280 reference	Clause	CAs		End entities		
				C(Root)	*C (Sub-CA)*	*C(TLS)*	*C (Key Agree)*	*C (Data Sign)*
1	AuthorityKeyIdentifier	4.2.1.1	5.6.4.4.2	o	m	m	m	m
2	SubjectkeyIdentifier	4.2.1.2	5.6.4.4.3	m	m	o	o	o
3	KeyUsage	4.2.1.3	5.6.4.4.4	m	m	m	m	m
4	CertificatePolicies	4.2.1.4	5.6.4.4.5	o	m	m	o	o
5	SubjectAltNames	4.2.1.6	5.6.4.4.6	o	o	o	o	o
6	IssuerAltNames	4.2.1.7	5.6.4.4.7	o	o	x	x	x
7	BasicConstraints	4.2.1.9	5.6.4.4.8	m	m	x	x	x
8	ExtendedKeyUsage	4.2.1.12	5.6.4.4.9	x	x	m	x	x
9	cRLDistributionPoints	4.2.1.13	5.6.4.4.10	o	o	x	x	x

C(Root): Certificate of the Root CA

C(Sub-CA): Certificate of a Sub-CA

C(TLS): Certificate for Transport Layer Security

C(KeyAgree): Certificate an ECDH capable key establishment key

C(DataSign): Certificate of an ECDSA capable signing key

5.6.4.4.2 Authority Key Identifier

- Extension-ID (OID): 2.5.29.35;
- Critical: FALSE;
- Description: the AuthorityKeyIdentifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate;
- Value: the `AuthorityKeyIdentifier` extension MUST include the `keyIdentifier` field.

The value of the `keyIdentifier` field needs to be computed either:

- with the method 1 defined in RFC 5280, 4.2.1.2 i.e. the `keyIdentifier` is composed of the 160-bit SHA-1 hash of the value of the BIT STRING `SubjectPublicKey` (excluding the tag, length, and number of unused bits); or
- with the method 2 defined in RFC 5280, 4.2.1.2 i.e. the keyIdentifier is composed of a four-bit type field with the value 0100 followed by the least significant 60 bits of the SHA-1 hash of the value of the BIT STRING `subjectPublicKey` (excluding the tag, length, and number of unused bits).

NOTE The choice of the method is left to project specific companion specifications.

5.6.4.4.3 SubjectKeyIdentifier

- Extension-ID (OID): 2.5.29.14;
- Critical: FALSE;
- Description: the `SubjectKeyIdentifier` extension provides a means of identifying certificates that contain a particular public key;
- Value: the `SubjectKeyIdentifier` extension MUST include the `keyIdentifier` field.

For the method of calculating the `keyIdentifier` see 5.6.4.4.2.

5.6.4.4.4 KeyUsage

- Extension-ID (OID): 2.5.29.15;
- Critical: TRUE;
- Description: the KeyUsage extension defines the purpose of the key contained in the certificate;
- Value: The bits that shall be set are shown in Table 19.

Table 19 – Key Usage extensions

Certificate	*C(Root)*	*C(Sub-CA)*	*C(TLS)*	*C (KeyAgree)*	*C (DataSign)*
Bits to be set	keyCertSign, cRLSign	keyCertSign, cRLSign	digitalSignature keyAgreement	keyAgreement	digitalSignature

For details, see RFC 5280, 4.2.1.3 and NSA3.

5.6.4.4.5 CertificatePolicies

- Extension-ID (OID): 2.5.29.32;
- Critical: FALSE;
- Description: the certificate policies extension contains a sequence of one or more policy information terms, each of which consists of an object identifier (OID) and optional qualifier. For details, see RFC 5280, 4.2.1.4;
- Value: contains the OID for the applicable certificate policy.

5.6.4.4.6 SubjectAltNames

- Extension-ID (OID): 2.5.29.17;
- Critical: TRUE if the “subject” field of the certificate is empty (an empty sequence), else FALSE.

Description: this extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate. If the subject name is an empty sequence, then the `subjectAltName` extension MUST be added in the End Entity Signature and Key Establishment Certificates and MUST be marked as critical. The `subjectAltName` extension is OPTIONAL otherwise and if included, MUST be marked as critical.

The `SubjectAltName` extension when used shall contain a single GeneralName of type OtherName that is further sub-typed as a HardwareModuleName (id-on-HardwareModuleName) as defined in RFC 4108. The hwSerialNum field shall be set to the system title.

- Value: See Table 20.

Table 20 – Subject Alternative Name values

Certificate	*C(Root)*	*C(Sub-CA)*	*C(TLS)*	*C (KeyAgree)*	*C (DataSign)*
rfc822Name	`<E-Mail Address>`	`<E-Mail Address>`	-	-	-
uRI	`<Web site>`	`<Web site>`	-	-	-
otherName	-	-	-	`<otherName>`	`<otherName>`
NOTE Values within the less-than – greater-than signs “< >” are to be assigned by the PKI or the CA as applicable.					

5.6.4.4.7 IssuerAltName

- Extension-ID (OID): 2.5.29.18;
- Critical: FALSE;
- Description: this extension is used to associate Internet style identities with the certificate issuer. For details see RFC 5280, 4.2.1.7;
- Value: See Table 21.

Table 21 – Issuer Alternative Name values

Certificate	*C(Root)*	*C(Sub-CA)*	*C(TLS)*	*C (KeyAgree)*	*C (dataSign)*
rfc822Name	<E-Mail Address>	<E-Mail Address>	–	–	–
URI	<Web site>	<Web site>	–	–	–
NOTE Values within the less-than – greater-than signs “< >” are to be assigned by the PKI or the CA as applicable.					

5.6.4.4.8 Basic constraints

- Extension-ID (OID): 2.5.29.19;
- Critical: TRUE;
- Description: the basic constraints extension identifies whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate. See RFC 5280, 4.2.1.9;
- Value: See Table 22.

Table 22 – Basic constraints extension values

Certificate	*C(Root)*	*C(Sub-CA)*	*C(TLS)*	*C (KeyAgree)*	*C (dataSign)*
cA	TRUE	TRUE	–	–	–
pathLenConstraint	See Note 1	See Note 1	–	–	–
NOTE 1 The value of the –optional – pathLenConstraint depends on the structure of the PKI.					

5.6.4.4.9 Extended Key Usage

- Extension-ID (OID): 2.5.29.37;
- Critical: FALSE;
- Description: Indicates that a certificate can be used as an TLS server certificate;
  - TLS server authentication OID: 1.3.6.1.5.5.7.3.1;
  - TLS client authentication OID: 1.3.6.1.5.5.7.3.2.

5.6.4.4.10 cRLDistributionPoints

- Extension-ID (OID): 2.5.29.31;
- Critical: FALSE;
- Description: The CRL distribution points extension identifies how CRL information is obtained;
- This extension is not used in DLMS®/COSEM server certificates.

5.6.4.4.11 Other extensions

All other extensions not described in this profile should be considered OPTIONAL; their inclusion or exclusion and their values will depend upon the particular application or PKI profile.

5.6.5 Suite B end entity certificate types to be supported by DLMS®/COSEM servers

Every DLMS®/COSEM server must use X.509 v3 format and contain either:

- a P-256 or P-384 ECDSA-capable signing key; or
- a P-256 or P-384 ECDH-capable key agreement key.

Every certificate must be signed using ECDSA. The signing CA's key must be P-256 or P-384 if the certificate contains a key on P-256. The signing CA's key must be P-384 if the certificate contains a key on P-384.

Depending on the security policy, the following X.509 v3 certificates shall be handled by DLMS®/COSEM end entities, see Table 23.

Table 23 – Certificates handled by DLMS®/COSEM end entities

Security suite 1	Security suite 2	Role
Root Certification Authority (Root-CA) Self-Signed Certificate using P-256 signed with P-256	Root Certification Authority (Root-CA) Self-Signed Certificate using P-384 signed with P-384	Trust anchor; there may be more than one.
Subordinate CA Certificate (Sub-CA) using P-256 signed with P-256	Subordinate CA Certificate (Sub-CA) using P-384 signed with P-384	Certificate of an issuing CA. Subordinate CAs may be also used as trust anchors.
–	Subordinate CA Certificate (Sub-CA) using P-256 signed with P-384	
End-Entity Signature Certificate using P-256 signed with P-256	End-Entity Signature Certificate using P-384 signed with P-384	Public key for ECDSA signature generation and verification
–	End-Entity Signature Certificate using P-256 signed with P-384	
End-Entity Key Establishment Certificate using P-256 signed with P-256	End-Entity Key Establishment Certificate using P-384 signed with P-384	Used with the One-Pass Diffie-Hellman C(1e, 1s) scheme or with the Static Unified Model C(0e, 2s, ECC CDH) scheme
–	End-Entity Key Establishment Certificate using P-256 signed with P-384	
End-Entity TLS Certificate using P-256 signed with P-256	End-Entity TLS Certificate using P-384 signed with P-384	
–	End-Entity TLS Certificate using P-256 signed with P-384	

An example Certificate is given in Annex H.

5.6.6 Management of certificates

5.6.6.1 Overview

This subclause 5.6.6 applies only to the management of public key certificates in DLMS®/COSEM servers, including:

- provisioning the server with trust anchors, see 5.6.6.2;
- provisioning the server with further CA certificates, see 5.6.6.3;
- security personalisation of the server, see 5.6.6.4;
- provisioning servers with certificates of clients and third parties, see 5.6.6.5;
- provisioning clients and third parties with certificates of servers, see 5.6.6.6;

- removing certificates, see 5.6.6.7.

NOTE Management of public key certificates in DLMS®/COSEM clients and in third party systems is out of the Scope of this International Standard.

5.6.6.2 Provisioning servers with trust anchors

Before starting steady state operations, servers shall be provisioned with trust anchors that will be used to validate the certificates. Trust anchors may be Root-CA (i.e. self-signed) certificates, Sub-CA certificates or directly trusted CA keys. A server may be provisioned with more than one trust anchor.

Trust anchors shall be placed in the server out of band (OOB).

Trust anchor certificates are stored together with other certificates.

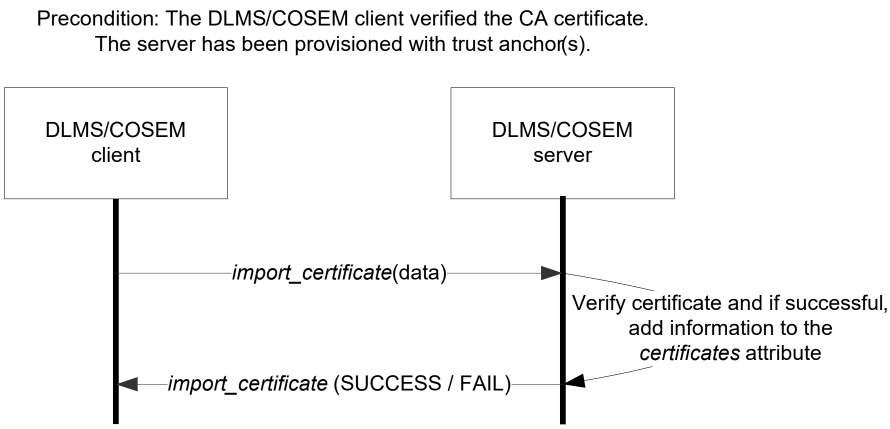
They can be exported, but they cannot be imported or removed.

Directly trusted CA keys cannot be exported.

5.6.6.3 Provisioning the server with further CA certificates

The server may be provisioned with further CA certificates that will be used to verify digital signatures on end device certificates.

For this purpose the *import_certificate* method of the “Security setup” object is available. The process is shown in Figure 24.



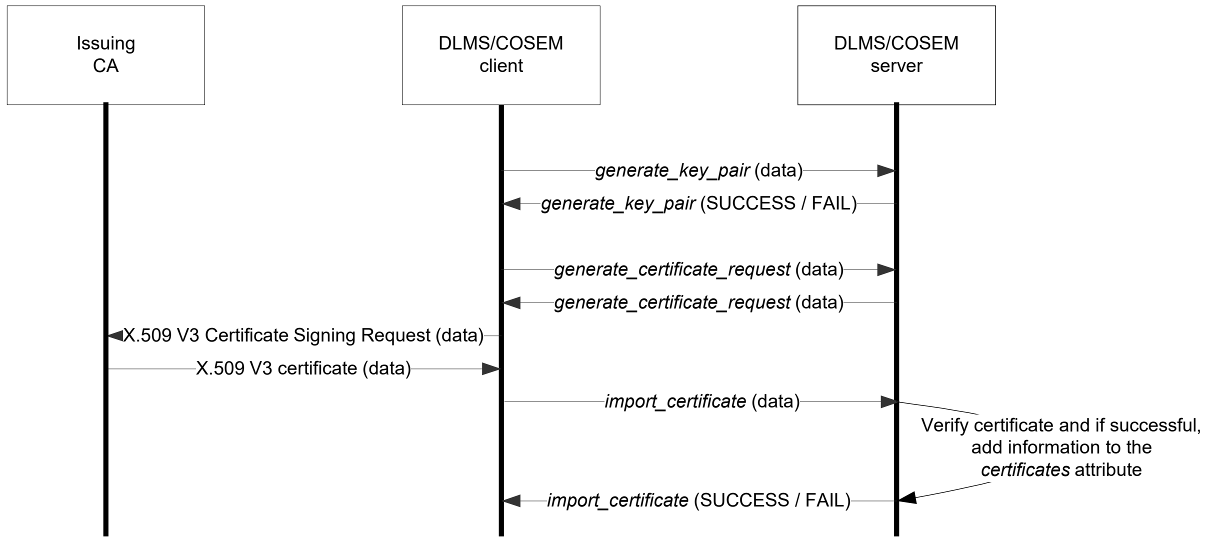
NOTE When a third party is responsible for managing CA certificates, then the *import_certificate* method may be invoked by that third party via the client acting as a broker.

Figure 24 – MSC for provisioning the server with CA certificates

5.6.6.4 Security personalisation of the server

Security personalisation means the provision of the server with asymmetric key pairs and the corresponding public key certificates. This can take place either:

- using the security primitives provided by the manufacturer to inject the private key and the public key certificates. The private keys have to be securely stored in the server and shall never be exposed; or
- using the appropriate methods of a “Security setup” object. This process can be used during the manufacturing process and whenever a new key pair and the related public key certificate need to be generated. This process is shown in Figure 25 and described below.



NOTE The security personalisation may be carried out by a third party instead of the client. In that case, the methods of the “Security setup” object may be invoked by that third party via the client acting as a broker.

Figure 25 – MSC for security personalisation of the server

- Step 1: the client invokes the *generate_key_pair* method. The method invocation parameters specify the key pair to be generated: digital signature, key agreement or TLS key pair;

  NOTE 1 The new key pair can be used in transactions once its certificate will have been imported and successfully verified.

- Step 2: the client invokes the *generate_certificate_request* method. The method invocation parameters identify the key pair for which the Certificate Signing Request (CSR) will be generated. The return parameters include the X.509 v3 CSR, signed by the private key of the newly generated key pair;
- Step 3: The client sends the X.509 v3 CSR to the CA. This message shall encapsulate the return parameters resulting from the invocation of the *generate_certificate_request* method. The CA – provided that the necessary conditions are met – issues the certificate and sends it to the client;

  NOTE 2 The format of the messages between the client and the issuing CA is out of the Scope of this International Standard.

- Step 4: The client invokes the *import_certificate* method. The method invocation parameters contain the certificate. The server verifies the certificate and if successful adds information on the certificate to the *certificates* attribute. If the verification fails, the certificate shall be discarded.

There may be only one key pair and certificate present for the same purpose (digital signature, key agreement, TLS). Therefore when the new certificate is successfully imported the old certificate is removed. From this point, the new key pair can be used for transactions.

For the details of method invocation and return parameters, see IEC 62056-6-2:2021, 4.4.7.

Parties using the server’s certificates can obtain these either:

- out of band;
- using the *export_certificate* method of the “Security setup” objects, see 5.6.6.6; or
- as part of the AARE (during HLS authentication), see 5.7.4.

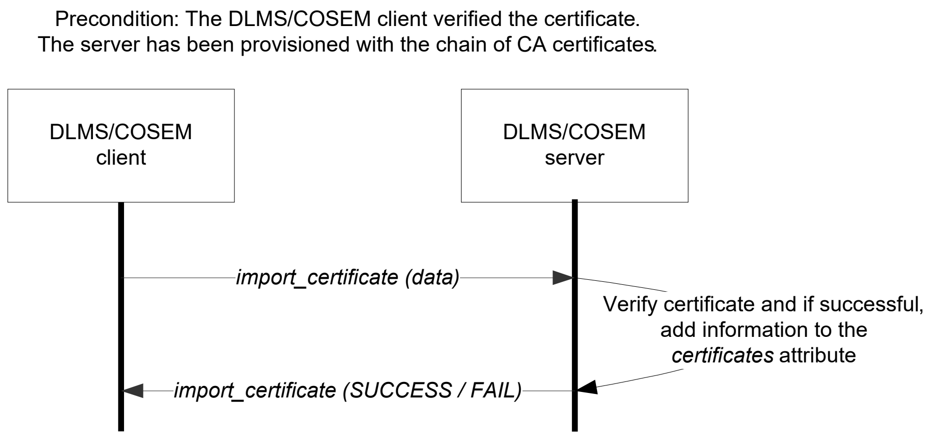
5.6.6.5 Provisioning servers with certificates of clients and third parties

To verify digital signatures, to perform key agreement using a scheme that uses static key agreement keys, or to establish a TLS connection, the server needs to have the appropriate public key certificates of the other party.

If, at the time of manufacturing, the client and/or third parties are already known, their public keys certificates may be injected into the server by the manufacturer.

NOTE Distribution of public key certificates to the manufacturer is out of Scope of this International Standard.

Otherwise, the servers can be provisioned with certificates of clients and third parties using the *import_certificate* method of the “Security setup” object. The method invocation parameter is the certificate to be placed in the server. For the details of method invocation and return parameters, see IIEC 62056-6-2:2021, 4.4.7. The process is shown in Figure 26.



NOTE The *import_certificate* (data) method may be also invoked by a third party, using the client as a broker.

Figure 26 – Provisioning the server with the certificate of the client

In the case of HLS authentication mechanism using ECDSA, the public key certificate of the clients’ digital signature key can be carried by the calling-AE-qualifier field of the AARQ. See 5.7.4.

5.6.6.6 Provisioning clients and third parties with certificates of servers

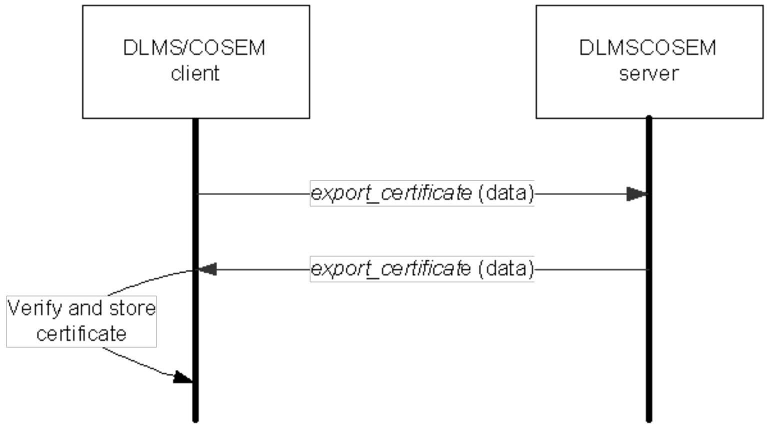
To verify digital signatures applied by the server, to perform key agreement that involves a static key agreement key, or to establish a TLS connection the client or third party needs to have the appropriate public key certificate of the server.

The certificate may be delivered with the server and inserted in clients / third parties OOB.

Alternatively, the client or third party can request the certificate from the server using the *export_certificate* method of the “Security setup” object. The method invocation parameter identifies the certificate requested.

NOTE In the case of HLS authentication using ECDSA – this is authentication_mechanism 7 – the public key certificate of the server for digital signature is transported in the AARE.

The return parameters – in the case of success – include the certificate. For the details of method invocation and return parameters, see IIEC 62056-6-2:2021, 4.4.7. The process is shown in Figure 27.



NOTE The *export_certificate* (data) method may be also invoked by a third party, using the client as a broker.

Figure 27 – Provisioning the client / third party with a certificate of the server

5.6.6.7 Certificate removal from the server

It is sometimes necessary to remove a public key certificate stored by the server.

NOTE 1 This may relate to certificates that belong to the server or certificates that belong to a client or third party.

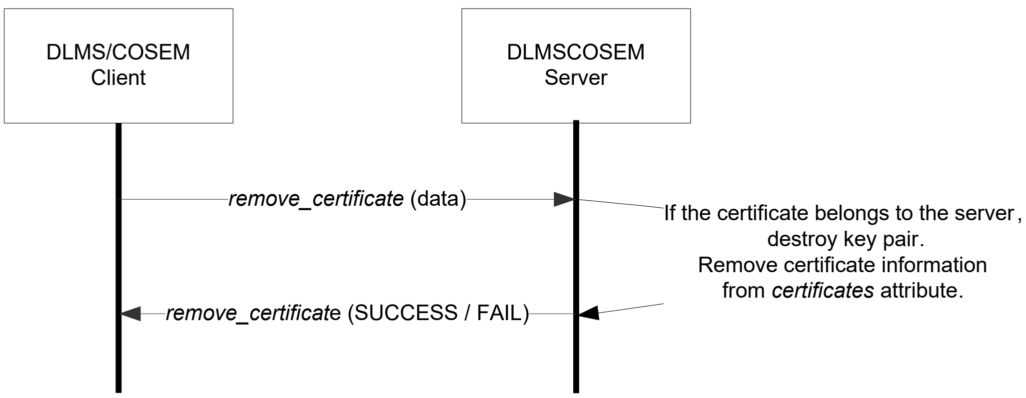
NOTE 2 The conditions when removal of a public key certificate is necessary are out of the Scope of this International Standard.

When a certificate that belongs to the server is removed, the private key associated with the public key shall be destroyed.

The information on the certificate removed shall be also removed from the *certificates* attribute of the "Security setup" object.

The key pair the public key certificate of which has been removed cannot be used any more for transactions.

The process is shown in Figure 28.



NOTE The *remove_certificate* (data) method may be also invoked by a third party, using the client as a broker.

Figure 28 – Remove certificate from the server

5.7 Applying cryptographic protection

5.7.1 Overview

The cryptographic algorithms specified in 5.3 can be applied:

- to protect the xDLMS APDUs see 5.7.2;
- to process the challenges during HLS authentication, see 5.7.4;
- to protect COSEM data, see 5.7.5.

5.7.2 Protecting xDLMS APDUs

5.7.2.1 Overview

This subclause 5.7.2 specifies how the cryptographic algorithms specified in 5.3.3 and 5.3.4 can be used to protect xDLMS APDUs:

- 5.7.2.2 specifies the possible values of security policy and access rights;
- 5.7.2.3 presents the types of ciphered APDUs;
- 5.7.2.4 specifies the use of the AES-GCM algorithm for authentication and encryption;
- 5.7.2.5 specifies the use of the ECDSA algorithm for digital signature.

When a COSEM object attribute or method related xDLMS service primitive is invoked by the AP, the service parameters include the Security_Options parameter. This parameter informs the AL on the kind of ciphered APDU to be used, on the protection to be applied, and includes the necessary security material. The AL builds first the APDU corresponding to the service primitive then it builds the ciphered APDU.

When the AL receives a ciphered APDU from a remote partner, it deciphers it and restores the original, unciphered APDU. When this is successfully done, it invokes the appropriate service primitive. The additional service parameters include the Security_Status and the Protection_Element parameters that inform the AP about the kind of ciphered APDU used, on the protection that has been verified and removed, and may include security material. See also 6.5.

5.7.2.2 Security policy and access rights values

In the case of “Security setup” version 1 – see IEC 62056-6-2:2021, 4.4.7 – the enum type shall be interpreted as an unsigned 8 type. The meaning of each bit is as shown in Table 24.

Table 24 – Security policy values (“Security setup” version 1)

Bit	Security policy
0	unused, shall be set to 0
1	unused, shall be set to 0
2	authenticated request
3	encrypted request
4	digitally signed request
5	authenticated response
6	encrypted response
7	digitally signed response
NOTE In the case of “Security policy” version 0 the possible security policy values are specified in IEC 62056-6-2:2021, 5.3.8. For both “Security policy” version 0 and 1 the value (0) means that no cryptographic protection is required.	

Access rights are held by the *object_list* attribute of “Association LN” or the *access_rights_list* of “Association SN” objects. The access_mode element of the access_rights determines the access kind and stipulates the cryptographic protection. It is an *enum* data type.

In the case of “Association LN” version 3 and “Association SN” version 4 – see IEC 62056-6-2:2021, 4.4.4 and 4.4.3 – the *enum* value is interpreted as an unsigned8: The meaning of each bit is as shown in Table 25.

For older versions, see their specification in IEC 62056-6-2:2021.

Table 25 – Access rights values (“Association LN” ver 3 “Association SN” ver 4)

Bit	Attribute access	Method access
0	read-access	access
1	write-access	not-used
2	authenticated request	authenticated request
3	encrypted request	encrypted request
4	digitally signed request	digitally signed request
5	authenticated response	authenticated response
6	encrypted response	encrypted response
7	digitally signed response	digitally signed response
Examples	enum (3): read-write enum (6) write access with authenticated request enum (255): read-write access with authenticated, encrypted and digitally signed request and response	enum (1): access enum (2): not used enum (5): access with authenticated request enum (253): access with authenticated, encrypted and digitally signed request and response

Access rights to COSEM object attributes and/or methods may require authenticated, encrypted and / or signed xDLMS APDUs. For this reason, APDUs with more protection than required by the security policy are always allowed. APDUs with less protection than required by the security policy and the access rights shall be rejected.

More protection in this context means that more kinds of protection are applied on the xDLMS APDU than what is requested by the security policy: for example, security policy requires that all APDUs are authenticated but access rights require that the APDU is encrypted and authenticated i.e. a higher protection.

5.7.2.3 Ciphered xDLMS APDUs

The different kind of ciphered xDLMS APDUs are shown in Table 26. See also 6.5. Ciphered xDLMS APDUs can be used in a ciphered application context only. On the other hand, in a ciphered application context, both ciphered and unciphered APDUs may be used.

Table 26 – Ciphered xDLMS APDUs

APDU type	Parties	Type of ciphering	Security services	Key used	Com-pression
Service-specific glo-ciphering or ded-ciphering	Client – Server	Symmetric key	Authentication Encryption	Block cipher key: - Dedicated key [1] - Global unicast / broadcast key *established* [2] outside the exchange [3], *identified* by the SC byte Authentication key: global, *established* [2] outside the exchange [3]	–
general-glo-ciphering general-ded-ciphering					Yes [5]
general-ciphering	Third party or Client – Server	Symmetric key	Authentication Encryption	Block cipher key: - Global unicast / broadcast, established [2] outside the exchange [3], identified as part of the exchange, or - Established [2] as part of the exchange [4] Authentication key: global, established [2] outside the exchange [3]	Yes [5]
general-signing		Asymmetric key	Digital signature	Signing key	No

1) Transported by the AARQ;

2) Key establishment may be key wrapping or key agreement;

3) In the server, these keys are held by the Security setup objects;

4) Key data is transported in the protected APDU;

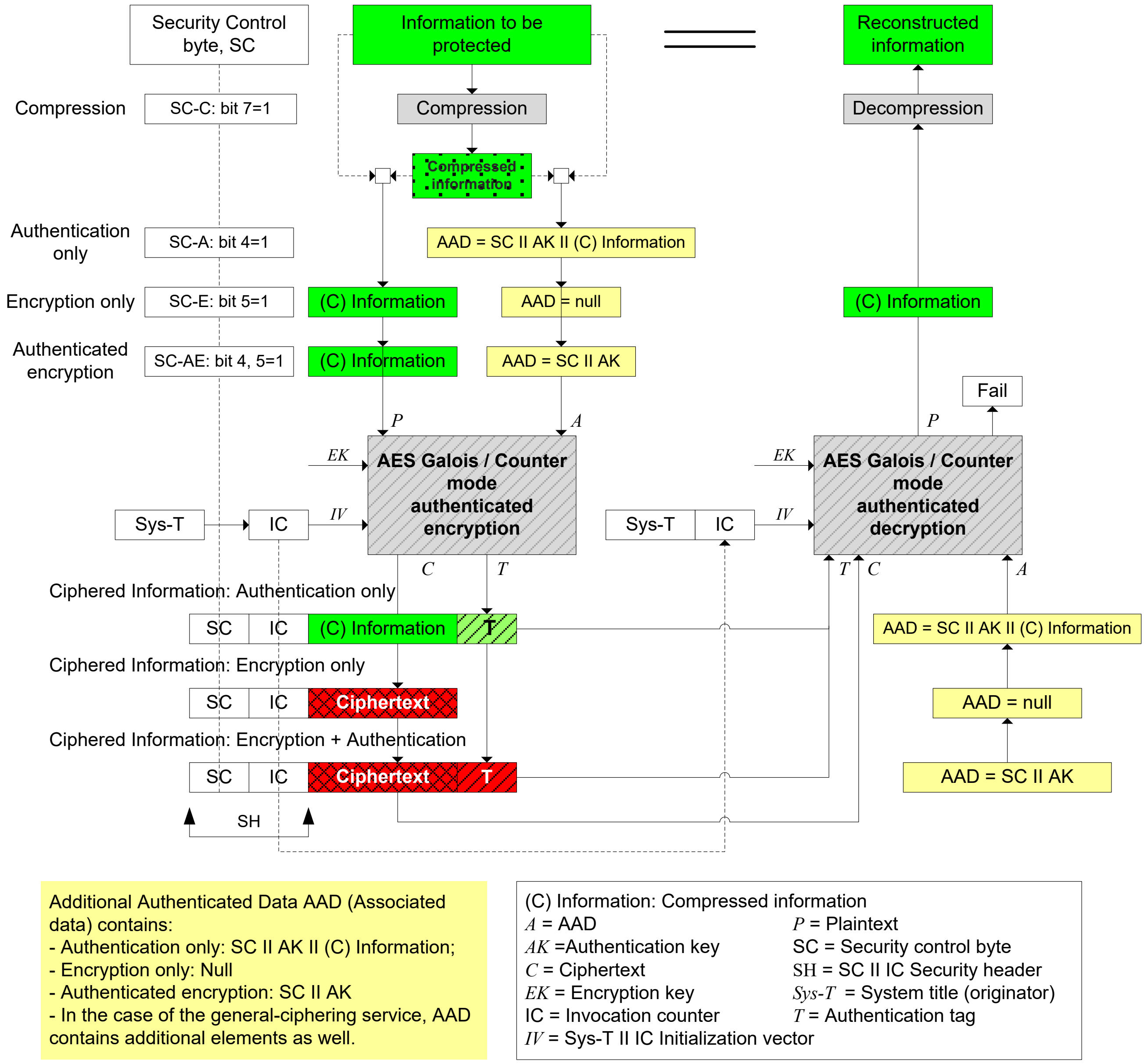
5) The use of compression is controlled by the Security Control byte.

5.7.2.4 Encryption, authentication and compression

5.7.2.4.1 Overview

Encryption and authentication to protect information using the AES-GCM algorithm is shown in Figure 29. See also 5.3.3.7. This algorithm can be combined with compression.

In the case of message protection, the information to be protected is an xDLMS APDU. In the case of COSEM data protection, the information to be protected is COSEM data, i.e. COSEM attribute value(s) or method invocation / return parameter(s).



NOTE In the case of general-ciphering, AAD also includes additional fields, see Table 28.

Figure 29 – Cryptographic protection of information using AES-GCM

The security material required is specified in 5.7.2.4.2 – 5.7.2.4.5.

5.7.2.4.2 The security header

The security header SH includes the security control byte concatenated with the invocation counter: SH = SC II IC. The security control byte is shown in Table 27 where:

- Bit 3…0: Security_Suite_Id, see 5.3.7;
- Bit 4: “A” subfield: indicates that authentication is applied;
- Bit 5: “E” subfield: indicates that encryption is applied;
- Bit 6: Key_Set subfield: 0 = Unicast,
  1 = Broadcast;
- Bit 7: Indicates the use of compression.

Table 27 – Security control byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3..0
Compression	Key_Set	E	A	Security_Suite_Id
The Key_Set bit is not relevant and shall be set to 0 when the service-specific dedicated ciphering, the general-ded-ciphering or the general-ciphering APDUs are used.				

5.7.2.4.3 Plaintext and Additional Authenticated Data

The plaintext denoted *P* and the Additional Authenticated Data denoted *A* depend on the kind of protection. They are shown in Table 28, where *SC* is the security control byte, *AK* is the authentication key and *I* is the information, i.e. the xDLMS APDU or the COSEM data to be protected.

Table 28 – Plaintext and Additional Authenticated Data

Security control, *SC*		Protection	*P*	*A, Additional Authenticated Data*	
E field	**A field**			**Service-specific glo-ciphering** **Service-specific ded-ciphering** **general-glo-ciphering** **general-ded-ciphering**	**general-ciphering**
0	0	None	–	–	–
0	1	Authenticated only	–	*SC* ∥ *AK* ∥ *(C) I*	*SC* ∥ *AK* ∥ transaction-id ∥ [1] originator-system-title ∥ [1] recipient-system-title ∥ [1] date-time ∥ [1] other-information ∥ [1] *(C) I*
1	0	Encrypted only	*(C) I*	–	–
1	1	Encrypted and authenticated	*(C) I*	*SC* ∥ *AK*	*SC* ∥ *AK* ∥ transaction-id ∥ [1] originator-system-title ∥ [1] recipient-system-title ∥ [1] date-time ∥ [1] other-information [1]
1) The fields transaction-id ….other-information are A-XDR encoded OCTET STRINGs. The length and the value of each field is included in the AAD.					

5.7.2.4.4 Encryption key and authentication key

These keys used by AES-GCM are specified in 5.3.3.7.4 and 5.3.3.7.5 respectively. The various keys used in DLMS®/COSEM and their establishment are specified in 5.5.

5.7.2.4.5 Initialization vector

See 5.3.3.7.3.

5.7.2.4.6 Service-specific ciphering xDLMS APDUs

For certain xDLMS APDUs – see 4.2.4.4.6 and 7.3.13 – a ciphered variant using the global key and one using the dedicated key is available. These ciphered APDUs can be used between a client and a server. The structure of the service-specific ciphering APDUs is shown in Figure 30. See also Table 28 and Table 29.

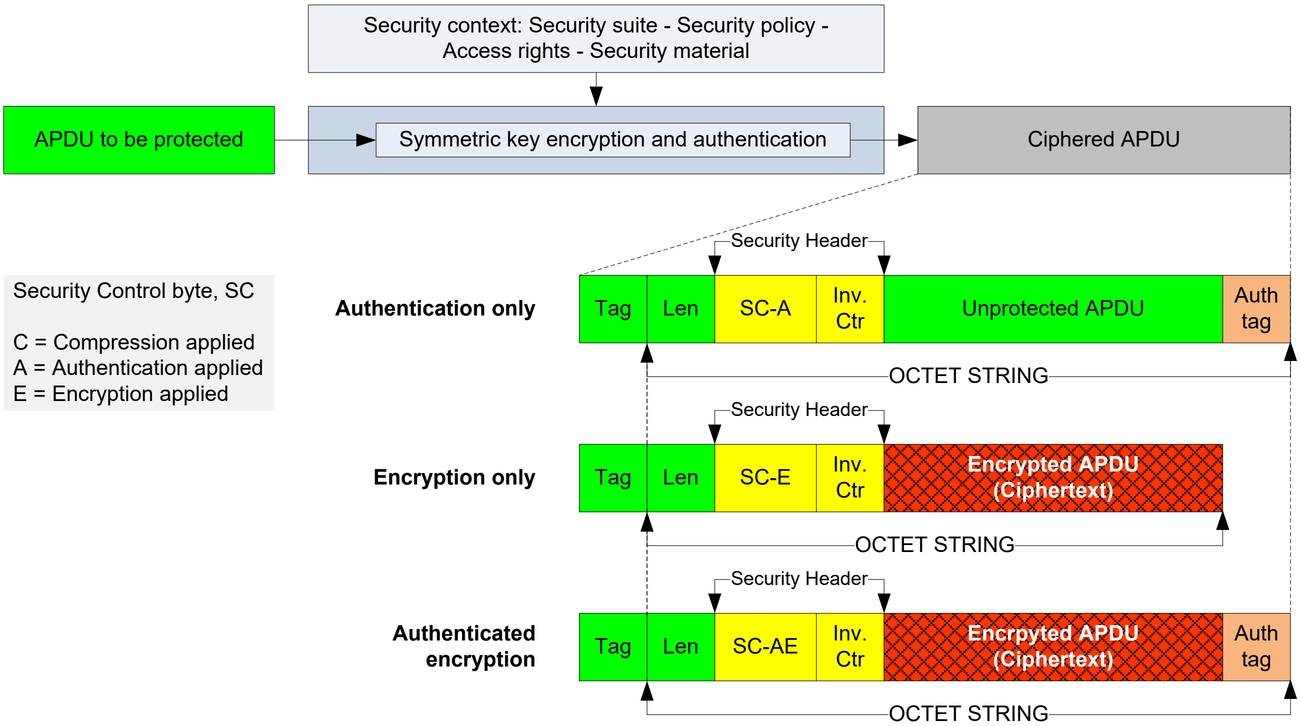


Figure 30 – Structure of service-specific global / dedicated ciphering xDLMS APDUs

5.7.2.4.7 The general-glo-ciphering and general-ded-ciphering xDLMS APDUs

These APDUs can be used to cipher other xDLMS APDUs using the global key or the dedicated key. They can be used between a client and a server. Their structure is shown in Figure 31. See also Table 28 and Table 29.

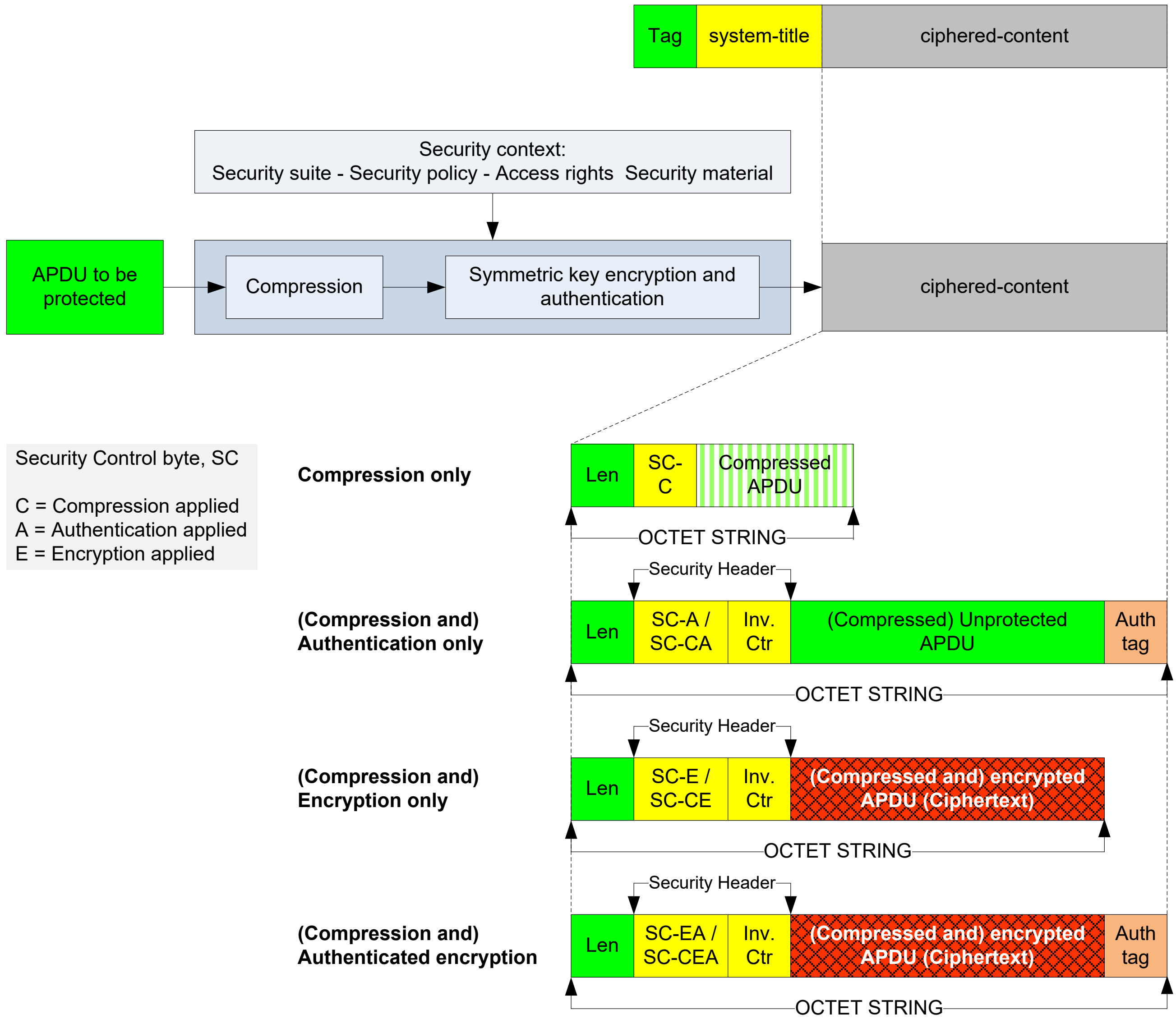
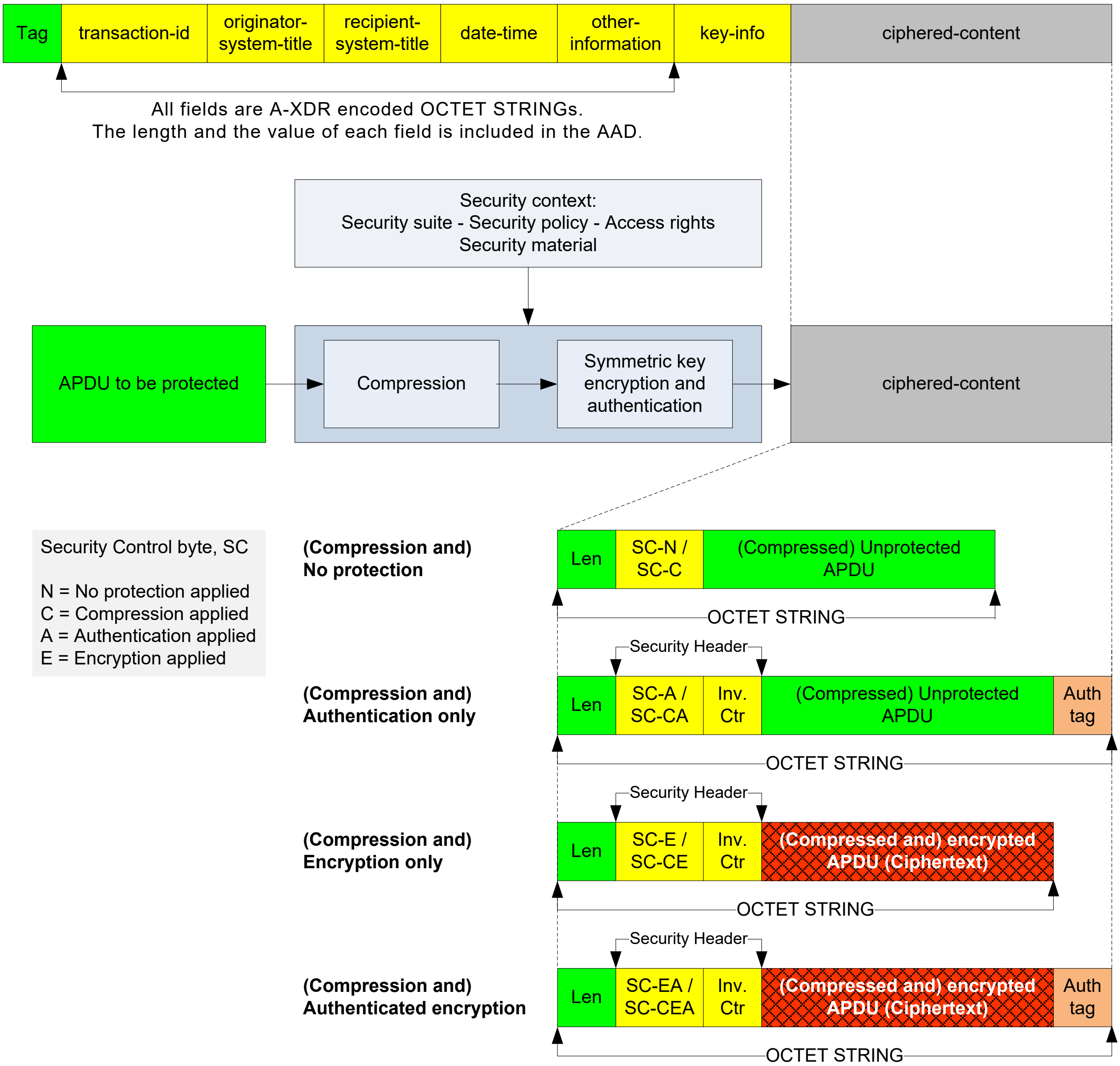


Figure 31 – Structure of general-glo-ciphering and general-ded-ciphering xDLMS APDUs

3466 ##### 5.7.2.4.8 The general-ciphering APDU

3467 The general-ciphering APDU can be used between a client and a server or between a third
3468 party and the server. These APDUs carry also the necessary information on the key to be used.
3469 The structure of the general-ciphering APDU is shown in Figure 32. See also Table 28 and
3470 Table 29.



3471

3472 **Figure 32 – Structure of general-ciphering xDLMS APDUs**

3473 ##### 5.7.2.4.9 Use of the fields of the ciphering xDLMS APDUs

3474 The use of the fields of the ciphering xDLMS APDUs is summarized in Table 29.

Table 29 – Use of the fields of the ciphering xDLMS APDUs

APDU field	Service-specific global / dedicated ciphering	general-glo-/ general-ded-ciphering	general-ciphering	Meaning
tag	Service specific	[219] [220]	[221]	The tag of the ciphering APDU; see 7.3.13
system-title	–	+	–	
transaction-id	–	–	+	
originator-system-title	–	–	+	
recipient-system-title	–	–	+	See 6.5.
date-time	–	–	+	
other-information	–	–	+	
key-info	–	–	+	
security control byte	+	+	+	Provides information on the protection applied, the key-set and the security suite used. See Table 27. [1)]
Invocation counter	+	+	+	The invocation field of the initialization vector. It is an integer counter which increments upon each invocation of the authenticated encryption function using the same key. When a new key is established the related invocation counter shall be reset to 0.
unprotected APDU	+	+	+	The unprotected APDU (same as the APDU to be protected).
encrypted APDU	+	+	+	The encrypted APDU i.e. the ciphertext.
authentication tag	+	+	+	Calculated by the AES-GCM algorithm, see 5.3.3.7.

[1)] In the case of the general-ciphering APDU, the key-set bit of the security control byte is not relevant and shall be set to zero.

5.7.2.4.10 Encoding example: global-get-request xDLMS APDU

Table 30 shows an encoding example of a service-specific global ciphering xDLMS APDU: glo-get-request.

3479 **Table 30 – Example: glo-get-request xDLMS APDU**

	X	**Contents**			***LEN (X)*** bytes	***Len (X)*** bits
Security material						
Security suite		GCM-AES-128				
System Title	*Sys-T*	**4D4D4D**0000BC614E (here, the five last octets contain the manufacturing number in hexa)			8	64
Invocation counter	*IC*	01234567			4	32
Initialization Vector	*IV*	*Sys-T* ∥ IC 4D4D4D0000BC614E01234567			12	96
Block cipher key (global)	*EK*	000102030405060708090A0B0C0D0E0F			16	128
Authentication Key	*AK*	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF			16	128
Security applied		**Authentication**	**Encryption**	**Authenticated encryption**		
Security control byte (with unicast key)	*SC*	*SC-A*	*SC-E*	*SC-AE*	1	8
		10	20	30		
Security header	*SH*	*SH* = *SC-A* ∥ IC	*SH* = *SC-E* ∥ IC	*SH* = *SC-AE* ∥ IC		
		1001234567	2001234567	3001234567	5	40
Inputs		**Authentication**	**Encryption**	**Authenticated encryption**		
xDLMS APDU to be protected	*APDU*	C0010000080000010000FF0200 (Get-request, attribute 2 of the Clock object)			13	104
Plaintext	*P*	Null	C0010000080000010000FF0200	C0010000080000010000FF0200	13	104
Associated data	*A*	*SC* ∥ *AK* ∥ *APDU*	–	*SC* ∥ *AK*		
Associated Data – Authentication	*A-A*	10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDFC0010000080000010000FF0200	–	–	30	240
Associated Data – Encryption	*A-E*	–	–	–	0	0
Associated Data – Authenticated encryption	*A-AE*	–	–	30D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	17	136
Outputs		**Authentication**	**Encryption**	**Authenticated encryption**		
Ciphertext	*C*	NULL	411312FF935A47566827C467BC	411312FF935A47566827C467BC	13	104
Authentication tag	*T*	06725D910F9221D263877516	–	7D825C3BE4A77C3FCC056B6B	12	96
The complete Ciphered APDU		*TAG* ∥ *LEN* ∥ *SH* ∥ *APDU* ∥ *T*	*TAG* ∥ *LEN* ∥ *SH* ∥ *C*	*TAG* ∥ *LEN* ∥ *SH* ∥ *C* ∥ *T*	–	–
Authenticated APDU		C81E1001234567C0010000080000010000FF020006725D910F9221D263877516	–	–	32	256
Encrypted APDU		–	C81220012345674411312FF935A47566827C467BC	–	20	160

	X	Contents			*LEN (X)* bytes	*Len (X)* bits
Authenticated and encrypted APDU		–	–	C81E30012345674 11312FF935A4756 6827C467BC7D825 C3BE4A77C3FCC05 6B6B	32	256
NOTE In this example the value of the invocation counter is 01234567. In the real life, the value shall be incremented with each invocation of the AES-GCM algorithm.						

Table 31 shows an example where the ACCESS.request and ACCESS.response APDUs shown in Table F. 10 are protected using authenticated encryption. The general-ciphering APDU specified in 5.7.2.4.8 is used. The encryption key is agreed on using the One-Pass Diffie-Hellman
C(1e, 1s, ECC CDH) key agreement scheme, see 5.3.4.6.3. The authentication key is the same as in Table 30.

Table 31 – ACCESS service with general-ciphering, One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) key agreement scheme

Message Elements	Contents	LEN (Bytes)
General-Ciphering	DD	1
transaction-id		0
length	08	1
value	0102030405060708	8
originator-system-title		0
length	08	1
value	4D4D4D0000BC614E	8
recipient-system-title		0
length	08	1
value	4D4D4D0000000001	8
date-time		0
length	00	1
value		0
other-information		0
length	00	1
value		0
key-info OPTIONAL	01	1
agreed-key CHOICE	02	1
key-parameters		0

length	01	1
value	01	1
key-ciphered-data		0
length	8180	2
value	C323C2BD45711DE4688637D919F92E9DB8 FB2DFC213A88D21C9DC8DCBA917D817051 1DE1BADB360D50058F794B0960AE11FA28 D392CFF907A62D13E3357B1DC0B51BE089 D0B682863B2217201E73A1A9031968A9B4 121DCBC3281A69739AF87429F5B3AC5471 E7B6A04A2C0F2F8A25FD772A317DF97FC5 463FEAC248EB8AB8BE	128
ciphered-content		0
length	81EB	2
value ACCESS.request with authenticated encryption SC II IC II ciphertext II auth. Tag	3100000000F435069679270C5BF4425EE5 777402A6C8D51C620EED52DBB188378B83 6E2857D5C053E6DDF27FA87409AEF502CD 9618AE47017C010224FD109CC0BEB21E74 2D44AB40CD11908743EC90EC8C40E221D5 17F72228E1A26E827F43DC18ED27B5F458 D66508B05A2A4CC6FED178C881AFC3BC67 064689BE8BB41C80ABB3C114A31F4CB03B 8B64C7E0B4CE77B2399C93347858888F92 239713B38DF01C4858245827A92EF33417 2EA636B31CBBDF2A96AD5D035F66AA38F1 A2D97D4BBA99622E6B5F18789CECB2DFB3 937D9F3E17F8B472098E6563238F375283 74809836002AEA6E7012D2ADFAA7	235
general-ciphering(encoded)	DD080102030405060708084D4D4D0000BC 614E084D4D4D00000000001000001020101 8180C323C2BD45711DE4688637D919F92E 9DB8FB2DFC213A88D21C9DC8DCBA917D81 70511DE1BADB360D50058F794B0960AE11 FA28D392CFF907A62D13E3357B1DC0B51B E089D0B682863B2217201E73A1A9031968 A9B4121DCBC3281A69739AF87429F5B3AC 5471E7B6A04A2C0F2F8A25FD772A317DF9 7FC5463FEAC248EB8AB8BE81EB31000000 00F435069679270C5BF4425EE5777402A6 C8D51C620EED52DBB188378B836E2857D5 C053E6DDF27FA87409AEF502CD9618AE47 017C010224FD109CC0BEB21E742D44AB40 CD11908743EC90EC8C40E221D517F72228 E1A26E827F43DC18ED27B5F458D66508B0 5A2A4CC6FED178C881AFC3BC67064689BE 8BB41C80ABB3C114A31F4CB03B8B64C7E0 B4CE77B2399C93347858888F92239713B3 8DF01C4858245827A92EF334172EA636B3 1CBBDF2A96AD5D035F66AA38F1A2D97D4B BA99622E6B5F18789CECB2DFB3937D9F3E	401

	17F8B472098E6563238F37528374809836 002AEA6E7012D2ADFAA7	
General-Ciphering	DD	1
transaction-id		0
length	08	1
value	0123456789012345	8
originator-system-title		0
length	08	1
value	4D4D4D0000000001	8
recipient-system-title		0
length	08	1
value	4D4D4D0000BC614E	8
date-time OPTIONAL		0
length	00	1
value		0
other-information		0
length	00	1
value		0
key-info OPTIONAL	01	1
agreed-key CHOICE	02	1
key-parameters		0
length	01	1

value	01	1
key-ciphered-data		0
length	8180	2
value	6439724714B47CD9CB988897D8424AB946 DCD083D37A954637616011B9C237877329 5F0F850D8DAFD1BBE9FE666E53E4F097CD 10B38B69622152724A90987444E1FF4797 4A1F6931A6502F58147463F0E8CC517D47 F55B0AC56DD8AC5C9D0E481934F2D90F98 93016BD82B6E3FFE21FF1588F3278B4E9D 98EB4FB62ADD64B380	128
ciphered-content		0
length	3D	1
value ACCESS.response with authenticated encryption SC II IC II ciphertext II auth. tag	3100000000B3FFCAA594642D8319CEC6B2 A233E2BF4621D6991B97E4565B986E8CCB E9A299D8E7869723638FF6BB20E66E175E 6F2D762CFD26B3D58733	61
general-ciphering (encoded)	DD080123456789012345084D4D4D000000 0001084D4D4D0000BC614E000001020101 81806439724714B47CD9CB988897D8424A B946DCD083D37A954637616011B9C23787 73295F0F850D8DAFD1BBE9FE666E53E4F0 97CD10B38B69622152724A90987444E1FF 47974A1F6931A6502F58147463F0E8CC51 7D47F55B0AC56DD8AC5C9D0E481934F2D9 0F9893016BD82B6E3FFE21FF1588F3278B 4E9D98EB4FB62ADD64B3803D3100000000 B3FFCAA594642D8319CEC6B2A233E2BF46 21D6991B97E4565B986E8CCBE9A299D8E7 869723638FF6BB20E66E175E6F2D762CFD 26B3D58733	226

5.7.2.5 Digital signature

The algorithm is the elliptic curve digital signature algorithm (ECDSA) as specified in 5.3.4.5.

The structure of the general-signing APDU is shown in Figure 33. For the additional fields, see Table 29 and 6.5.

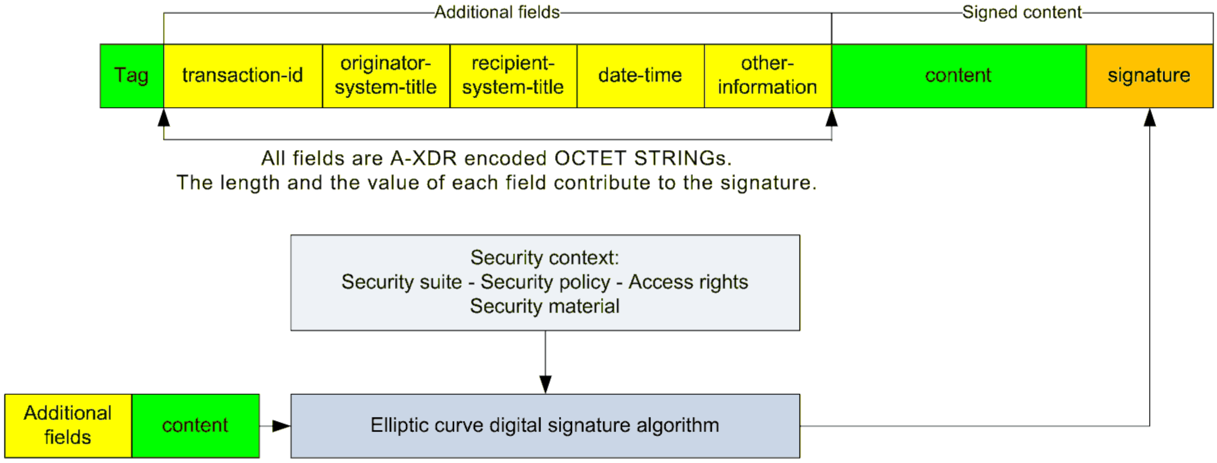


Figure 33 – Structure of general-signing APDUs

5.7.3 Multi-layer protection by multiple parties

Cryptographic protection can be applied by multiple parties. Generally the parties are:

- a server;
- a client;
- a third party.

Each party can apply one or multiple layers of protection:

- to apply encryption, authentication or authenticated encryption, the ciphering APDUs are used. A third party shall use the general-ciphering APDU. The client can use any of the ciphering APDUs. Authenticated encryption is considered to be a single layer of protection;
- to apply digital signature, the general-signing APDU is used.

If both ciphering and digital signature is applied by the same party for the same party, then normally the digital signature is applied first.

Both the general-ciphering and general-signing APDUs include the Originator_System_Title and the Recipient_System_Title, identifying the party that applied the protection and the party that shall check / remove the protection.

The protection to be applied on the response depends on the security policy and the access rights on the response and on the protection applied on the request. If a kind of protection has been applied on the request by a party, then the same kind of protection will be applied for the same party in the response. However if a kind of protection which was applied on the request is not required on the response, than no protection will be applied on the response for that party.

Example 1 If the request was digitally signed by the third party and authenticated by the client, and the required protection on the response is authentication and digital signature, then the response will be authenticated for the client and digitally signed for the third party.

Example 2 If the request was digitally signed by the third party and authenticated by the client, and the required protection on the response is authentication only, then the response will be authenticated for the client and no protection will be applied to the third party. (The TP will receive a general-ciphering APDU without any protection applied.)

If a protection is required on the response that was not applied on the request, then the server cannot determine from the request which party has the response to be protected for. Therefore it shall apply the protection for all parties.

See also Annex J.

5.7.4 HLS authentication mechanisms

HLS authentication requires cryptographic processing of the challenges exchanged by the client and the server. The HLS authentication mechanisms, the information exchanged and the formulae to process the challenges are shown in Table 32.

Table 32 – DLMS®/COSEM HLS authentication mechanisms

Authentication mechanism	**Pass 1:** C →S	**Pass 2:** S →C	**Pass 3:** C →S f(StoC)	**Pass 4:** S→C f(CtoS)
	Carried by			
	AARQ	AARE	XX.request reply_to_HLS authentication	XX.response reply_to_HLS authentication
mechanism_id(2) HLS man. Spec.	*CtoS*: Random string 8-64 octets	*StoC*: Random string 8-64 octets	Man. Spec.	Man. Spec.
mechanism_id(3) HLS MD5 [1]			**MD5**(StoC || HLS Secret)	**MD5**(CtoS || HLS Secret)
mechanism_id(4) HLS SHA-1 [1]			**SHA-1**(StoC || HLS Secret)	**SHA-1**(CtoS || HLS Secret)
mechanism_id(5) HLS GMAC	*CtoS*: Random string 8-64 octets Optionally: System-Title-C in calling-AP-title	*StoC*: Random string 8-64 octets Optionally: System-Title-S in responding-AP-title	SC II IC II **GMAC** (SC || AK || StoC)	SC II IC II **GMAC** (SC || AK || CtoS)
mechanism_id(6) HLS SHA-256			**SHA-256** (HLS_Secret || SystemTitle-C || SystemTitle-S || StoC II CtoS)	**SHA-256** (HLS_Secret || SystemTitle-S || SystemTitle-C || CtoS || StoC)
mechanism_id(7) HLS ECDSA	*CtoS*: Random string 32 to 64 octets Optionally: System-Title-C in calling-AP-title, Cert-Sign-Client in calling-AE-qualifier	*StoC*: Random string 32 to 64 octets Optionally: System-Title-S in responding-AP-title, Cert-Sign-Server responding-AE-qualifier	**ECDSA**(SystemTitle-C || SystemTitle-S || StoC II CtoS)	**ECDSA**(SystemTitle-S || SystemTitle-C || CtoS II StoC)
Legend: - C: Client, S: Server, CtoS: Challenge client to server, StoC: Challenge server to client - IC: Invocation counter - xx.request / .response: xDLMS service primitives used to access the *reply_to_HLS authentication* method of the “Association SN / LN” object.				
[1] The use of authentication mechanisms 3 and 4 are not recommended for new implementations.				
NOTE The system titles and the Certificates have to be sent only if not already known by the other party.				

Where the system titles and the certificates for the digital signature key are also needed, these may be transported in the AARQ / AARE APDUs, carrying the COSEM-OPEN service .request / .response, see Figure 12. The System_Title and Cert-Sign may be already known; in this case they do not have to be transported. If these elements are not available, the result of the processing of the challenge fails and the AA shall not be established.

Table 33 provides a test vector for HLS authentication-mechanism 5 with GMAC.

Table 33 – HLS example using authentication-mechanism 5 with GMAC

Security material	X	Contents		LEN(*X*) bytes	len(*X*) bits
Security suite		GCM-AES-128			
System Title	*Sys-T*	Client	Server		
		4D4D4D0000000001	**4D4D4D**0000BC614E		
		(here, the five last octets contain the manufacturing number in hexa)		8	64
Invocation counter	*IC*	00000001	01234567	4	32
Initialization Vector	*IV*	*Sys-T* II IC		12	96
		Client	Server		
		4D4D4D0000000001 00000001	4D4D4D0000BC614E 01234567		
Block cipher key (global)	*EK*	000102030405060708090A0B0C0D0E0F		16	128
Authentication Key	*AK*	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF		16	128
Security control byte	SC	10		1	8
		Pass 1: Client sends challenge to server			
CtoS		4B35366956616759 “K56iVagY”		8	64
		Pass 2: Server sends challenge to client			
StoC		503677524A323146 “P6wRJ21F”		8	64
		Pass 3: Client processes StoC			
SC II AK II StoC		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF503677 524A323146			
T = **GMAC** (SC II AK II StoC)		1A52FE7DD3E72748973C1E28		12	96
f(StoC) = SC II IC II T		10000000011A52FE7DD3E72748973C1E28		17	136
		Pass 4: Server processes CtoS			
SC II AK II CtoS		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF4B3536 6956616759			
T (SC II AK II CtoS)		FE1466AFB3DBCD4F9389E2B7		12	96
f(CtoS) = SC II IC II T		1001234567FE1466AFB3DBCD4F9389E2B7		17	136

Table 34 provides a test vector for HLS authentication-mechanism 7 with ECDSA.

Table 34 – HLS example using authentication-mechanism 7 with ECDSA

<table>
<tr><th>Security Material</th><th>X</th><th>Contents</th><th>LEN (Bytes)</th></tr>
<tr><td>Security Suite</td><td></td><td>ECDH-ECDSA-AES-128-GCM</td><td></td></tr>
<tr><td>Curve</td><td></td><td>P-256</td><td></td></tr>
<tr><td>Domain Parameters</td><td>D</td><td>See Table G. 1.</td><td></td></tr>
<tr><td>System Title Client</td><td>Sys-TC</td><td>4D4D4D0000BC614E</td><td>8</td></tr>
<tr><td>System Title Server</td><td>Sys-TS</td><td>4D4D4D0000000001</td><td>8</td></tr>
<tr><td>Private Key Client</td><td>Pri-KC</td><td>E9A045346B2057F1820318AB125493E9AB36CE
590011C0FF30090858A118DD2E</td><td>32</td></tr>
<tr><td>Private Key Server</td><td>Pri-KS</td><td>B582D8C910018302BA3131BAB9BB6838108BB9
408C30B2E49285985256A59038</td><td>32</td></tr>
<tr><td>Public Key Client</td><td>Pub-KC</td><td>917DBFECA43307375247989F07CC23F53D4B96
3AF8026C749DB33852011056DFDBE8327BD69C
C149F018A8E446DDA6C55BCD78E596A56D4032
36233F93CC89B3</td><td>64</td></tr>
<tr><td>Public Key Server</td><td>Pub-KS</td><td>E4D07CEB0A5A6DA9D2228B054A1F5E295E1747
A963974AF75091A0B0BC2FB92DA7D2ABD9FDD4
1579F36A1C8171A0CB638221DF1949FD95C8FA
E148896920450D</td><td>64</td></tr>
<tr><td>Challenge Client To Server</td><td>CtoS</td><td>2CA1FC2DE9CD03B5E8E234CEA16F2853F6DC5F
54526F4F4995772A50FB7E63B3</td><td>32</td></tr>
<tr><td>Challenge Server To Client</td><td>StoC</td><td>18E95FFE3AD0DCABDC5D0D141DC987E270CB0A
395948D4231B09DE6579883657</td><td>32</td></tr>
<tr><td>ECDSA(SystemTitle-C || SystemTitle-S ||StoC || CtoS)
(calculated with Pri-KC)</td><td>f(StoC)</td><td>C5C6D6620BDB1A39FCE50F4D64F0DB712D6FB5
7A64030B0C297E1250DC859660D3B1FA334AD8
0411807369F5DD3BC17B59894C9E9C11C59376
580D15A2646D16</td><td>64</td></tr>
<tr><td>ECDSA(SystemTitle-S || SystemTitle-C || CtoS || StoC)
(calculated with Pri-KS)</td><td>f(CtoS)</td><td>946C2E3E4F18291571F4A45ACB708610057469
4A3BAF67D2D147FE8F92481A5AB2186C5CBC3F
80E94482D9388B85C6A73E5FD687F09773C1F6
15AA2A905ED057</td><td>64</td></tr>
<tr><td colspan="4">NOTE The values of the public keys are represented here as FE2OS(x_p)II FE2OS(y_p).</td></tr>
</table>

5.7.5 Protecting COSEM data

The cryptographic algorithms applied to xDLMS APDUs can be also applied to COSEM data, i.e. attribute values and method invocation / return parameters. This is achieved by accessing attributes and/or methods of other COSEM objects indirectly through instances of the “Data protection” interface class, see IEC 62056-6-2:2021, 4.4.9.

The list of data to be protected, the required protection and the protection parameters are determined by the “Data protection” objects.

“Data protection” objects allow applying or removing protection when reading or writing a list of attributes, or when invoking methods of COSEM objects. Protection to be applied / removed may include any combination of authentication, encryption and digital signature.

The APDUs carrying the service invocations to access the attributes and methods of “Data protection” objects are protected as required by the prevailing security policy and the access rights of the “Data protection” object.

6 DLMS®/COSEM application layer service specification

6.1 Service primitives and parameters

In general, the services of a layer (or sublayer) are the capabilities it offers to a user in the next higher layer (or sublayer). In order to provide its service, a layer builds its functions on the services it requires from the next lower layer. Figure 34 illustrates this notion of service hierarchy and shows the relationship of the two correspondent N-users and their associated N-layer peer protocol entities.

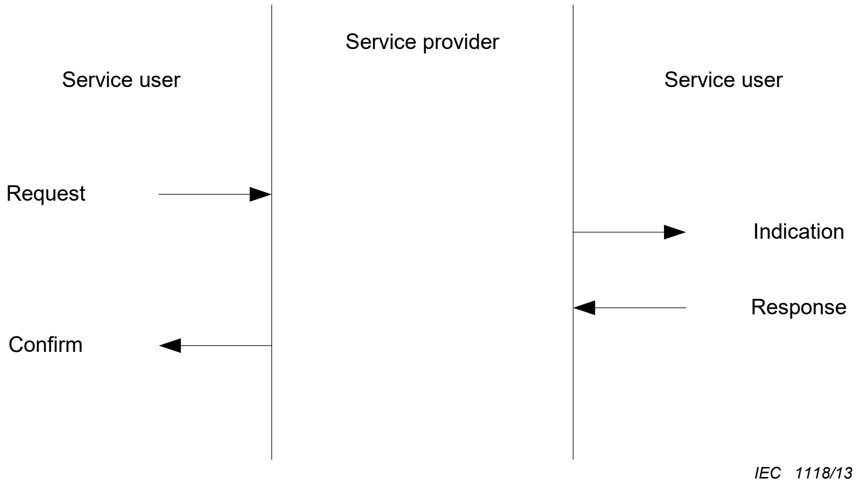


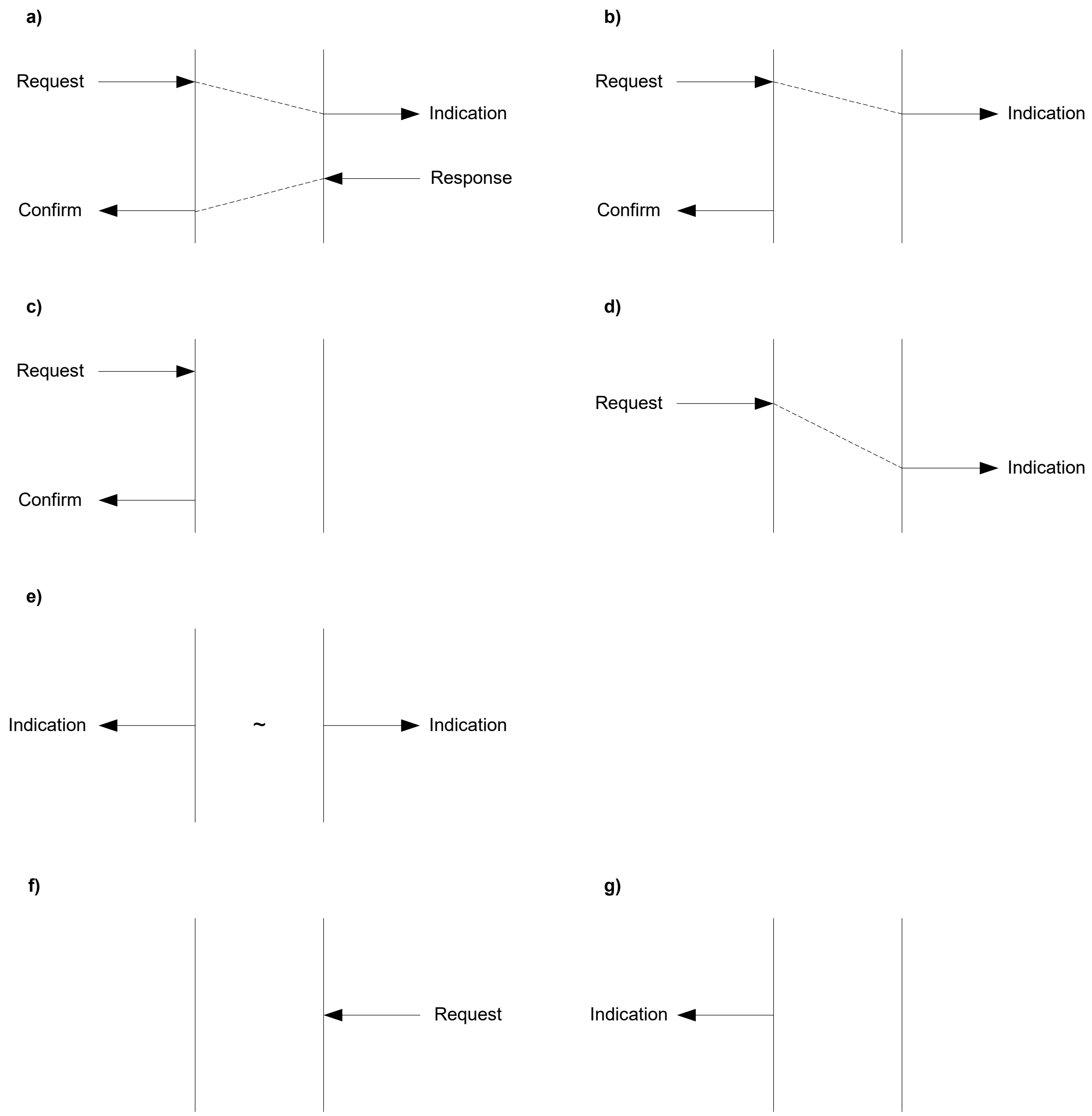
Figure 34 – Service primitives

Services are specified by describing the information flow between the N-user and the N-layer. This information flow is modelled by discrete, instantaneous events, which characterize the provision of a service. Each event consists of passing a service primitive from one layer to the other through an N-layer service access point associated with an N-user. Service primitives convey the information required in providing a particular service. These service primitives are an abstraction in that they specify only the service provided rather than the means by which the service is provided. This definition of service is independent of any particular interface implementation.

Services are specified by describing the service primitives and parameters that characterize each service. A service may have one or more related primitives that constitute the activity that is related to the particular service. Each service primitive may have zero or more parameters that convey the information required to provide the service. Primitives are of four generic types:

- REQUEST: The request primitive is passed from the N-user to the N-layer to request that a service be initiated;
- INDICATION: The indication primitive is passed from the N-layer to the N-user to indicate an internal N-layer event that is significant to the N-user. This event may be logically related to a remote service request, or may be caused by an event internal to the N-layer;
- RESPONSE: The response primitive is passed from the N-user to the N-layer to complete a procedure previously invoked by an indication primitive;
- CONFIRM: The confirm primitive is passed from the N-layer to the N-user to convey the results of one or more associated previous service request(s).

Possible relationships among primitive types are illustrated by the time-sequence diagrams shown in Figure 35. The figure also indicates the logical relationship of the primitive types. Primitive types that occur earlier in time and are connected by dotted lines in the diagrams are the logical antecedents of subsequent primitive types.



IEC 1119/13

Figure 35 – Time sequence diagrams

The service parameters of the COSEM AL service primitives are presented in a tabular format. Each table consists of two to five columns describing the service primitives and their parameters. In each table, one parameter – or a part of it – is listed on each line. In the appropriate service primitive columns, a code is used to specify the type of usage of the parameter. The codes used are listed in Table 35.

Some parameters may contain sub-parameters. These are indicated by labelling of the parameters as M, U, S or C, and indenting all sub-parameters under the parameter. Presence of the sub-parameters is always dependent on the presence of the parameter that they appear under. For example, an optional parameter may have sub-parameters; if the parameter is not supplied, then no sub-parameters may be supplied.

Table 35 – Codes for AL service parameters

M	The parameter is mandatory for the primitive.
U	The parameter is a user option, and may or may not be provided depending on dynamic usage by the ASE user.
S	The parameter is selected among other S-parameters as internal response of the server ASE environment.
C	The parameter is conditional upon other parameters or the environment of the ASE user.
(–)	The parameter is never present.
=	The " (=) " code following one of the M, U, S or C codes indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. For instance, an " M (=) " code in the .indication service primitive column and an "M" in the .request service primitive column means that the parameter in the .indication primitive is semantically equivalent to the one in the .request primitive.

Throughout this standard, the following rules are observed regarding the naming of terms:

- the name of ACSE services and the data transfer services using LN referencing is written in uppercase. Examples are: COSEM-OPEN, GET;
- the name of the data transfer services using SN referencing is written in title case. Examples are: Read, Write;
- camel notation is used in the following cases: DataNotification, EventNotification, TriggerEventNotificationSending, UnconfirmedWrite, InformationReport;
- the types of the LN service primitives may be mentioned in two alternative forms. Examples: “GET.request service primitive of Request_Type == NORMAL” or “GET-REQUEST-NORMAL service primitive”;
- service parameter name elements are capitalized and joined with an underscore to signify a single entity: Examples are Protocol_Connection_Parameters and COSEM_Attribute_Descriptor;
- when the same parameter may occur several times, this is indicated by repeating the parameter in curly brackets. Example: Data { Data };
- in the data transfer service specifications, parameters used with block transfer only are shown in bold. Example: **DataBlock_G**;
- direct reference to a service parameter uses the capitalized form, while indirect (non-specific) reference uses the normal text without underscore joining. A direct reference example is: “The COSEM_Attribute_Descriptor parameter references a COSEM object attribute.” An indirect (non-specific) reference example is: “A GET-REQUEST-NORMAL service primitive contains a single COSEM attribute descriptor”;
- the names of COSEM data transfer APDUs using LN referencing are capitalized and joined with a dash to signify a single entity. Example: Get-Request-Normal;
- the names of COSEM data transfer APDUs using SN referencing use the camel notation. Example: ReadRequest.

6.2 The COSEM-OPEN service

Function

The function of the COSEM-OPEN service is to establish an AA between peer COSEM APs. It uses the A-ASSOCIATE service of the ACSE. The COSEM-OPEN service provides only the framework for *transporting* this information. To provide and verify that information is the job of the appropriate COSEM AP.

Semantics of the service primitives

The COSEM-OPEN service primitives shall provide parameters as shown in Table 36.

Table 36 – Service parameters of the COSEM-OPEN service primitives

	.request	.indication	.response	.confirm
Protocol_Connection_Parameters	M	M (=)	M	M (=)
ACSE_Protocol_Version	U	U (=)	U	U (=)
Application_Context_Name	M	M (=)	M	M (=)
Called_AP_Title	U	U (=)	–	–
Called_AE_Qualifier	U	U(=)	–	–
Called_AP_Invocation_Identifier	U	U (=)	–	–
Called_AE_Invocation_Identifier	U	U (=)	–	–
Calling_AP_Title	C	C (=)	–	–
Calling_AE_Qualifier	U	U (=)	–	–
Calling_AP_Invocation_Identifier	U	U (=)	–	–
Calling_AE_Invocation_Identifier	U	U (=)	–	–
Local_Or_Remote	–	–	–	M
Result	–	–	M	M
Failure_Type	–	–	M	M
Responding_AP_Title	–	–	C	C (=)
Responding_AE_Qualifier	–	–	U	U (=)
Responding_AP_Invocation_Identifier	–	–	U	U (=)
Responding_AE_Invocation_Identifier	–	–	U	U (=)
ACSE_Requirements	U	U (=)	U	U (=)
Security_Mechanism_Name	C	C (=)	C	C (=)
Calling_Authentication_Value	C	C (=)	–	–
Responding_Authentication_Value	–	–	C	C (=)
Implementation_Information	U	U (=)	U	U (=)
Proposed_xDLMS_Context	M	M (=)	–	–
Dedicated_Key	C	C (=)	–	–
Response_Allowed	C	C (=)		
Proposed_DLMS_Version_Number	M	M (=)	–	–
Proposed_DLMS_Conformance	M	M (=)	–	–
Client_Max_Receive_PDU_Size	M	M (=)	–	–
Negotiated_xDLMS_Context	–	–	S	S (=)
Negotiated_DLMS_Version_Number	–	–	M	M (=)
Negotiated_DLMS_Conformance	–	–	M	M (=)
Server_Max_Receive_PDU_Size	–	–	M	M (=)
VAA_Name			M	M (=)
xDLMS_Initiate_Error			S	S (=)
User_Information	U	C (=)	–	–
Service_Class	M	M (=)	–	–

The service parameters of the COSEM-OPEN.request service primitive, except the Protocol_Connection_Parameters, the User_Information parameter and – depending on the communication profile – the Service_Class parameter are carried by the fields of the AARQ APDU sent by the client.

The service parameters of the COSEM-OPEN.response service primitive, except the Protocol_Connection_Parameters is carried by the fields of the AARE APDU sent by the server.

The A-ASSOCIATE service and the AARQ and AARE APDUs are specified in 7.2. Encoding examples are given in Annex D and Annex E.

The Protocol_Connection_Parameters parameter is mandatory. It contains all information necessary to use the layers of the communication profile, including the communication profile (protocol) identifier and the addresses required. It identifies the participants of the AA. The elements of this parameter are passed to the entities managing lower layer connections and to the lower layers as appropriate.

The ACSE_Protocol_Version parameter is optional. If present, the default value shall be used.

The Application_Context_Name parameter is mandatory. In the request primitive, it holds the value proposed by the client. In the response primitive, it holds the same value or the value supported by the server.

The use of the Called_AP_Title, Called_AE_Qualifier, Called_AP_Invocation _Identifier, Called_AE_Invocation_Identifier parameters is optional. Their use is not specified in this International Standard.

The use of the Calling_AP_Title parameter is conditional. When the Application_Context_Name indicates an application context using ciphering, it may carry the client system title specified in 4.1.3.4.

The use of the Calling_AE_Qualifier parameter is conditional. When the Application_Context_Name indicates an application context using ciphering, it may carry the public digital signature key certificate of the client.

The use of the Calling_AP_Invocation_Identifier is optional. Its use is not specified in this International Standard.

The use of the Calling_AE_Invocation_Identifier parameter is optional. When present, it carries the identifier of the client-side user of the AA.

NOTE 1 The client user identification mechanism is specified in IEC 62056-6-2:2021, 4.4.2.

The Local_or_Remote parameter is mandatory. It indicates the origin of the COSEM-OPEN.confirm primitive. It is set to Remote if the primitive has been generated following the reception of an AARE APDU from the server. It is set to Local if the primitive has been locally generated.

The Result parameter is mandatory. In the case of remote confirmation, it indicates whether the Server accepted the proposed AA or not. In the case of local confirmation, it indicates whether the client side protocol stack accepted the request or not.

The Failure_type parameter is mandatory. In the case of remote confirmation, it carries the information provided by the server. In the case of local and negative confirmation, it indicates the reason for the failure.

The use of the Responding_AP_Title parameter is conditional. When the Application_Context_Name parameter indicates an application context using ciphering, it may carry the server system title specified in 4.1.3.4.

The use of the Responding_AE_Qualifier is conditional. When the Application_Context_Name indicates an application context using ciphering, it may carry the public digital signature key certificate of the server.

The use of the Responding_AP_Invocation_Identifier and Responding_AE_Invocation_ Identifier parameters is optional. Their use is not specified in this International Standard.

The ACSE_Requirements parameter is optional. It is used to select the optional authentication functional unit of the A-Associate service for the association; see 7.2.1.

The presence of the ACSE_Requirements parameter depends on the authentication mechanism used:

- in the case of Lowest Level Security authentication it shall not be present; only the Kernel functional unit will be used;
- in the case of Low Level Security (LLS) authentication it shall be present in the .request primitive and it may be present in the .response service primitive and it shall indicate authentication (bit 0 set);
- in the case of High Level Security (HLS) authentication, it shall be present both in the .request and the .response service primitives and it shall indicate authentication (bit 0 set).

The Security_Mechanism_Name parameter is conditional. It is present only if the authentication functional unit has been selected. If present, the .request primitive holds the value proposed by the client and the .response primitive holds the value required by the server, i.e. the one to be used by the client.

The Calling_Authentication_Value parameter and the Responding_Authentication_Value parameters are conditional. They are present only if the authentication functional unit has been selected. They hold the client authentication value / server authentication value respectively, appropriate for the Security_Mechanism_Name.

The Implementation_Information parameter is optional. Its use is not specified in this international standard.

The Proposed_xDLMS_Context parameter holds the elements of the proposed xDLMS context carried by the xDLMS InitiateRequest APDU, placed in the user-information field of the AARQ APDU.

The Dedicated_Key element is conditional. It may be present only, when the Application_Context_Name parameter indicates an application context using ciphering. The dedicated key is used for dedicated ciphering of xDLMS APDUs exchanged within the AA established.

When the dedicated key is present, the xDLMS InitiateRequest APDU shall be authenticated and encrypted using the AES-GCM algorithm, the global unicast encryption key and the authentication key (if in use). In addition it shall also be digitally signed if required by the security policy.

The xDLMS InitiateRequest APDU shall be protected the same way as described above, when the dedicated key is not present, but it is necessary to protect the RLRQ APDU by including the protected xDLMS InitiateRequest in its user-information field. See 6.3.

The use of Response_Allowed element is conditional. It indicates if the server is allowed to respond with an AARE APDU, i.e. if the AA to be established is confirmed (Response_Allowed == TRUE) or not confirmed (Response_Allowed == FALSE).

The Proposed_DLMS_Version_Number element holds the proposed DLMS® version number; see 4.2.4.

The Proposed_DLMS_Conformance element holds the proposed conformance block; see 7.3.1.

The Client_Max_Receive_PDU_Size element holds the maximum length of the xDLMS APDUs the client can receive; see Table 2.

If the xDLMS context proposed by the client is acceptable for the server, then the response service primitive shall contain the Negotiated_xDLMS_Context parameter. It holds the elements of the negotiated xDLMS context, carried by the xDLMS InitiateResponse APDU, placed in the user-information field of the AARE APDU. If the xDLMS InitiateRequest APDU has been ciphered, the xDLMS InitiateResponse APDU shall be also ciphered the same way.

The Negotiated_DLMS_Version_Number element holds the negotiated DLMS® version number. See 4.2.4.

The Negotiated_DLMS_Conformance element holds the negotiated conformance block. See 7.3.1.

The Server_Max_Receive_PDU_Size element carries the maximum length of the xDLMS APDUs the server can receive; see Table 2.

The VAA_name element carries the dummy value of 0x0007 in the case of LN referencing, and the base_name of the current Association object, 0xFA00, in the case of SN referencing.

If the xDLMS context proposed by the client is not acceptable for the server, then the response service primitive shall carry the xDLMS_Initiate_Error parameter. It is carried by the ConfirmedServiceError APDU, with appropriate diagnostic elements, placed in the user-information field of the AARE APDU.

The User_Information parameter is optional. If present, it shall be passed on to the supporting layer, provided it is capable to carry it. The indication primitive shall then contain the user-specific information carried by the supporting lower protocol layer(s); see Annex A.

NOTE 2 The User_Information parameter of the COSEM-OPEN service is not to be confused with the user-information field of the AARQ / AARE APDUs.

The Service_Class parameter is mandatory. It indicates whether the service shall be invoked in a confirmed or in an unconfirmed manner. The handling of this parameter may depend on the communication profile; see Annex A.

Use

Possible logical sequences of the COSEM-OPEN service primitives are illustrated in Figure 35:

- for confirmed AA – successful or unsuccessful – establishment, item a);
- for unconfirmed AA establishment, item b);
- in the case of a pre-established AA or an unsuccessful attempt due to a local error, item c).

The .request primitive is invoked by the client AP to request the establishment of a confirmed or an unconfirmed AA with a server AP.

NOTE 3 Before the invocation of the COSEM-OPEN.request primitive, the physical layers shall be connected. Depending on the communication profile, the invocation of this primitive may also imply the connection of other lower layers.

Upon reception of the request invocation, the AL constructs and sends an AARQ APDU to the server.

The .indication primitive is generated by the server AL when a correctly formatted AARQ APDU is received.

The .response primitive is invoked by the server AP to indicate to the AL whether the proposed AA is accepted or not. It is invoked only if the proposed AA is confirmed. The AL constructs then an AARE APDU and sends it to its peer, containing the service parameters received from the AP.

The .confirm primitive is generated by the client AL to indicate to the client AP whether the AA previously requested is accepted or not:

- remotely, when an AARE APDU is received;
- locally, if the requested AA already exists; this includes pre-established AAs;
- locally, if the corresponding .request primitive has been invoked with Service_Class == Unconfirmed;
- locally, if the requested AA is not allowed;
- locally, if an error is detected: missing or not correct parameters, failure during the establishment of the requested lower layer connections, missing physical connection, etc.

The protocol for establishing an AA is specified in 7.2.4. Communication profile specific rules are specified in Annex A.

6.3 The COSEM-RELEASE service

Function

The function of the COSEM-RELEASE service is to gracefully release an existing AA. Depending on the way it is invoked, it uses the A-RELEASE service of the ACSE or not.

Semantics of the service primitives

The COSEM-RELEASE service primitives shall provide parameters as shown in Table 37.

Table 37 – Service parameters of the COSEM-RELEASE service primitives

	.request	.indication	.response	.confirm
Use_RLRQ_RLRE	U	C(=)	C(=)	-
Reason	U	U (=)	U	U (=)
Proposed_xDLMS_Context	C	C (=)	–	–
Negotiated_xDLMS_Context	–	–	C	C (=)
Local_Or_Remote	–	–	–	M
Result	–	–	M	M
Failure_Type	–	–	–	C
User_Information	U	C (=)	U	C (=)

The Use_RLRQ_RLRE parameter in the .request primitive is optional. If present, its value may be FALSE (default) or TRUE. It indicates whether the ACSE A-RELEASE service – involving an RLRQ / RLRE APDU exchange – should be used or not. The A-RELEASE service and the RLRQ / RLRE APDUs are specified in 7.2. The Use_RLRQ_RLRE parameter in the .response primitive is conditional. If it was present in the .indication primitive and if its value was TRUE, it shall also be present and its value shall be TRUE. Otherwise, it shall not be present or its value shall be FALSE.

If the value of the Use_RLRQ_RLRE parameter is FALSE, then the AA can be released by disconnecting the supporting layer of the AL.

The Reason parameter is optional. It may be present only if the value of the Use_RLRQ_RLRE is TRUE. It is carried by the reason field of the RLRQ / RLRE APDU respectively.

When used on the .request primitive, this parameter identifies the general level of urgency of the request. It takes one of the following symbolic values:

- normal;
- urgent (not available in DLMS®/COSEM); or
- user defined.

When used on the .response primitive, this parameter identifies information about why the acceptor accepted or rejected the release request. Note that in DLMS®/COSEM, the server cannot reject the release request. It takes one of the following symbolic values:

- normal;
- not finished; or
- user defined.

NOTE 1 The value “not finished” is used in the .response primitive when the acceptor is forced to release the association but wishes to give a warning that it has additional information to send or receive.

The Proposed_xDLMS_Context parameter is conditional. It is present only if the value of the Use_RLRQ_RLRE is TRUE and the AA to be released has been established with an application context using ciphering. This option allows securing the COSEM-RELEASE service, and avoiding thereby a denial-of-service attack that may be carried out by unauthorized releasing of the AA.

In the .request primitive, the Proposed_xDLMS_Context parameter shall be the same as in the COSEM-OPEN.request service primitive, having established the AA to be released. It is carried by the xDLMS InitiateRequest APDU, protected the same way as in the AARQ and placed in the user-information field of the RLRQ APDU.

If the xDLMS InitiateRequest APDU can be successfully deciphered, then the .response primitive shall carry the same Negotiated_xDLMS_Context parameter as in the COSEM-OPEN.response primitive. It is carried by the xDLMS InitiateResponse APDU, protected the same way as in the AARE and placed in the user-information field of the RLRE APDU.

Otherwise, the RLRQ APDU is silently discarded.

The Local_or_Remote parameter is mandatory. It indicates the origin of the COSEM-RELEASE.confirm primitive.

It is set to Remote if either:

- a RLRE APDU has been received from the server; or
- a disconnect confirmation service primitive has been received.

It is set to Local if the primitive has been locally generated.

The Result parameter is mandatory. In the .response primitive, it indicates whether the server AP can accept the request to release the AA or not. As servers cannot refuse such requests, its value should normally be SUCCESS unless the AA referenced does not exist.

The Failure_Type parameter is conditional. It is present if Result == ERROR. In this case, it indicates the reason for the failure. It is a locally generated parameter on the client side.

The User_Information parameter in the .request primitive is optional. If present, it is passed to the supporting layer, provided it is able to carry it. The .indication primitive contains then the user-specific information carried by the supporting lower protocol layer(s). Similarly, it is optional in the .response primitive. If present, it is passed to the supporting layer. In the .confirm primitive, it may be present only when the service is remotely confirmed. It contains then the user-specific information carried by the supporting lower protocol layer(s).

NOTE 2 The User_Information parameter of the COSEM-RELEASE service is not to be confused with the user-information field of the RLRQ / RLRE APDUs.

The specification of the content of the User_Information parameter is not within the scope of this international standard. See also Annex A.

Use

Possible logical sequences of the COSEM-RELEASE service primitives are illustrated in Figure 35:

- for successful release of a confirmed AA , item a);
- for release of an unconfirmed AA , item b); and
- for an unsuccessful attempt due to a local error, item c).

The use of the COSEM-RELEASE service depends on the value of the Use_RLRQ_RLRE parameter. When it is invoked with Use_RLRQ_RLRE == TRUE, the service is based on the ACSE A-RELEASE service. Otherwise, the invocation of the service leads to the disconnection of the supporting layer.

The .request primitive is invoked by the client AP to request the release of a confirmed or an unconfirmed AA with a server AP. Upon reception of the request invocation with Use_RLRQ_RLRE == TRUE, the AL constructs and sends an RLRQ APDU to the server. Otherwise, it sends an XX-DISCONNECT.request primitive (where XX is the supporting lower protocol layer).

The .indication primitive is generated by the server AL if:

- a RLRQ APDU is received. If a deciphering error occurs, the RLRQ APDU is silently discarded (no .indication primitive is generated); or
- an XX-DISCONNECT.request is received.

The .response primitive is invoked by the server AP, but only if the AA to be released is confirmed. Note, that the server AP cannot refuse this request. Upon reception of the .response service primitive, the server AL:

- sends a RLRE APDU, if the Use_RLRQ_RLRE parameter is TRUE; or
- sends an XX-DISCONNECT.response otherwise.

The .confirm primitive is generated by the client AL to indicate to the client AP whether the requested release of the AA is accepted or not:

- remotely, when an XX-DISCONNECT.cnf primitive is received. The supporting layer is disconnected; or
- remotely, when a RLRE APDU is received. The supporting layer is not disconnected; or
- locally, upon the expiry of a time-out on waiting for an RLRE APDU; or
- locally, when an RLRQ APDU to release an unconfirmed AA is sent out; or
- locally, when a local error is detected: missing or incorrect parameters, or communication failure at lower protocol layer level etc.

If the RLRE APDU received contains a ciphered xDLMS InitiateResponse APDU but it cannot be deciphered, then the RLRE APDU shall be discarded. It is left to the client to cope with the situation.

The protocol for releasing an AA is specified in 7.2.5. Communication profile specific rules are specified in Annex A. See also 7.2.1.

6.4 COSEM-ABORT service

Function

The function of the COSEM-ABORT service is to indicate an unsolicited disconnection of the supporting layer.

Semantics

The COSEM-ABORT service primitives shall provide parameters as shown in Table 38.

Table 38 – Service parameters of the COSEM-ABORT service primitives

	.indication
Diagnostics	U

The Diagnostics parameter is optional. It shall indicate the possible reason for the disconnection, and may carry lower protocol layer dependent information as well. Specification of the contents of this parameter is not within the scope of this international standard.

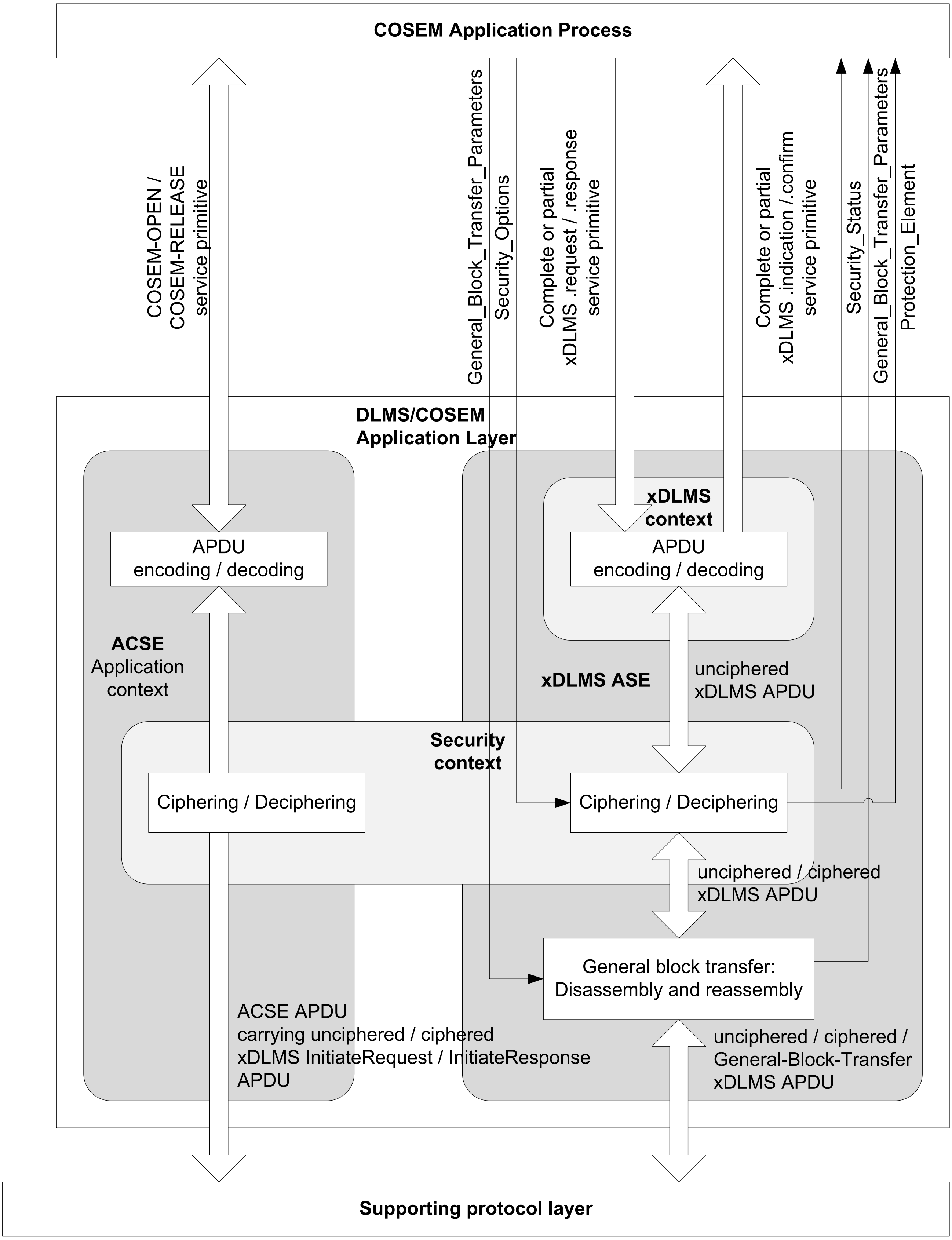
Use

The COSEM-ABORT.indication primitive is locally generated both on the client and on the server side to indicate to the COSEM AP that a lower layer connection closed in an unsolicited manner. The origin of such an event can be an external event (for example the physical line is broken), or an action of a supporting layer connection manager AP, present in some profiles, when the supporting layer connection is not managed by the COSEM AL. This shall cause the COSEM APs to abort any existing AAs, except the pre-established ones on the server side.

The protocol for the COSEM-ABORT service is specified in 7.2.5.3.

6.5 Protection and general block transfer parameters

To control cryptographic protection of xDLMS APDUs and the GBT mechanism, additional service parameters are passed between the AL and the AP as shown in Figure 36 and Table 39.



3918

3919 NOTE For services initiated by the client, the service primitives are .request, .indication, .response and .confirm.
3920 For unsolicited services – initiated by the server – the service primitives are .request and .indication.

3921 **Figure 36 – Additional service parameters to control cryptographic protection and GBT**

Table 39 – Additional service parameters

	.request	.indication	.response	.confirm
Additional_Service_Parameters	U	–	U (=)	
Invocation_Type	M	–	M (=)	–
Security_Options	C	–	C (=)	–
General_Block_Transfer_Parameters	C	–	C (=)	–
Block_Transfer_Streaming	M	–	M (=)	–
Block_Transfer_Window	M	–	M (=)	–
Service_Parameters	M	–	M (=)	–
Additional_Service_Parameters	–	U	–	U (=)
Invocation_Type	–	U	–	U (=)
Security_Status	–	C	–	C (=)
General_Block_Transfer_Parameters	–	C	–	C (=)
Block_Transfer_Window	–	M	–	M (=)
Service_Parameters	–	M	–	M (=)
Protection_Element	–	C	–	C (=)
NOTE The service primitives available depend on the kind of the service.				

The Additional_Service_Parameters are present only if ciphering or GBT is used.

The Invocation_Type parameter is mandatory: it indicates if the service invocation is complete or partial. Possible values: COMPLETE, FIRST-PART, ONE-PART and LAST-PART.

NOTE 1 Partial service invocations may be useful when the service parameters are long. However, there is no direct relationship between the partial service invocations and the general-block-transfer APDUs.

The Security_Options parameter is conditional: it is present only if the application context is a ciphered one, the .request / .response service primitive has to be ciphered and Invocation_Type = COMPLETE or FIRST-PART. It determines the protection to be applied by the AL. See also Table 40, 7.3.13 and Figure 59.

The General_Block_Transfer_Parameters parameter is conditional: it is present only if general block transfer (GBT) is used and Invocation_Type = COMPLETE or FIRST-PART. It provides information on the GBT streaming capabilities:

- the Block_Transfer_Streaming parameter is present only in .request and .response service primitives. It is passed by the AP to the AL to indicate if the AL is allowed to send general-block-transfer APDUs using streaming (TRUE) or not (FALSE);
- the Block_Transfer_Window parameter indicates the window size supported, i.e. the maximum number of blocks that can be received in a window.

The streaming process itself is managed by the AL. See 7.3.13.

The Service_Parameters are mandatory: they include the parameters of xDLMS service invocations. If Invocation_Type != COMPLETE, then it includes a part of the service parameters.

The Security_Status parameter is conditional: it is present only if cryptographic protection has been applied. It carries information on the protection that has been verified / removed by the AL. It may be present in all type of service invocations. See Table 40.

The Protection_Element parameter is conditional: it is present only if the APDU has been authenticated or signed. See Table 40.

Table 40 – Security parameters

	.request	.indication	.response	.confirm
Security_Options	C	–	C (=)	–
Security_Options_Element {Security_Options_Element}	M	–	M (=)	–
Security_Protection_Type	M	–	M (=)	–
Glo_Ciphering	S	–	S (=)	–
Ded_Ciphering	S	–	S (=)	–
General_Glo_Ciphering	S	–	S (=)	–
General_Ded_Ciphering	S	–	S (=)	–
General_Ciphering	S	–	S (=)	–
General_Signing	S	–	S (=)	–
With General_Glo_Ciphering and General_Ded_Ciphering				
System_Title	U	–	U (=)	–
With General_Ciphering and General_Signing				
Transaction_Id	U	–	U (=)	–
Originator_System_Title	U	–	U (=)	–
Recipient_System_Title	U	–	U (=)	–
Date_Time	U	–	U (=)	–
Other_Information	U	–	U (=)	–
With General_Ciphering				
Key_Info_Options	C	–	C (=)	–
Identified_Key_Options	S	–	S (=)	–
Wrapped_Key_Options	S	–	S (=)	–
Agreed_Key_Options	S	–	S (=)	–
With Glo_Ciphering, Ded_Ciphering, General_Glo_Ciphering, General_Ded_Ciphering, General_Ciphering				
Security_Control	M	–	M (=)	–

Table 40 *(continued)*

Security_Status	–	C	–	C (=)
Security_Status_Element {Security_Status_Element}	–	M	–	M (=)
Security_Protection_Type		M		M (=)
Glo_Ciphering	–	S	–	S (=)
Ded_Ciphering	–	S	–	S (=)
General_Glo_Ciphering	–	S	–	S (=)
General_Ded_Ciphering	–	S	–	S (=)
General_Ciphering	–	S	–	S (=)
General_Signing	–	S	–	S (=)
With General_Glo_Ciphering and General_Ded_Ciphering				
System_Title	–	U	–	U (=)
With General_Ciphering and General_Signing				
Transaction_Id	–	U	–	U (=)
Originator_System_Title	–	U	–	U (=)
Recipient_System_Title	–	U	–	U (=)
Date_Time	–	U	–	U (=)
Other_Information	–	U	–	U (=)
With General_Ciphering				
Key_Info_Status	–	C	–	C (=)
Identified_Key_Status	–	S	–	S (=)
Wrapped_Key_Status	–	S	–	S (=)
Agreed_Key_Status	–	S	–	S (=)
With Glo_Ciphering, Ded_Ciphering, General_Ded_Ciphering, General_Glo_Ciphering, General_Ciphering				
Security_Control	–	M	–	M (=)
The protection element is present when authentication or digital signature is applied.				
Protection_Element {Protection_Element}	–	C	–	C (=)
Invocation_Counter	–	C	–	C (=)
Authentication_Tag	–	C	–	C (=)
Signature	–	C	–	C (=)

The Security_Options parameter contains one Security_Options_Element parameter for each kind of protection to be applied. Similarly, the Security_Status parameter contains one Security_Status_Element parameter for each kind of protection that has been applied. See also 5.7.3.

The Security_Options_Element and Security_Status_Element parameters include the following sub-parameters:

- the Security_Protection_Type sub-parameter is mandatory: it identifies the ciphered APDU to be used; see Table 41;
- the System_Title subparameter is optional. When present, it holds the system title of the sender. It can be present only with General_Glo_Ciphering and General_Ded_Ciphering;

NOTE 2 The purpose to include system-title of the sender is to allow the other party to build the initialization vector where the system-title has not been exchanged during the media specific registration process or during the AARQ / AARE exchange.

Table 41 – APDUs used with security protection types

Security_Protection_Type	APDU
Glo_Ciphering	Service-specific glo-ciphering
Ded_Ciphering	Service-specific ded-ciphering
General_Glo_Ciphering	general-glo-ciphering
General_Ded_Ciphering	general-ded-ciphering
General_Ciphering	general-ciphering
General_Signing	general-signing
See also Table 26.	

The following five parameters are optionally present with General_Ciphering and General_Signing:

- Transaction_Id: identifies the transaction between two parties;
- Originator_System_Title: indicates the system title of the originator of the protected APDU;
- Recipient_System_Title: indicates the system title of the recipient, i.e. the entity which will verify / remove the protection that has been applied to the APDU. In the case of broadcast, the Recipient_System_Title shall be an empty string;
- The Date_Time parameter is optional. When present, it indicates the date and time of the invocation of the .request / .response service primitive. Unless otherwise specified in a project specific companion specification, the Date_Time parameter in the response shall be present if it was present in the request and shall not be present in the response if it was not present in the request;

  NOTE 3 If any of the four parameters above is not used, then an octet-string of length zero shall be included.

- the Other_Information parameter is optional. When present, it holds additional information concerning the protection. Its content may be specified in project specific companion specifications;

The Key_Info_Options parameter is conditional: when protection has to be applied, it carries information on the symmetric key that has been used by the originator / is to be used by the recipient. The key information is sent / received as part of the ciphered APDU:

- Identified_Key_Options (see 5.5.3): it can be used when the partners share the key; this may be the global unicast encryption key or the global broadcast encryption key;
- Wrapped_Key_Options (see 5.5.4): in this case, a wrapped key is sent;
- Agreed_Key_Options (see 5.5.5): in this case, the partners use a Diffie-Hellman key agreement scheme to agree on the key;

Security_Control: contains the Security Control byte, see Table 27.

The Protection_Element parameter is conditional: it shall be present if the APDU has been authenticated or digitally signed. It may be present in all type of service invocations, but it may be empty if it is not yet available (this may occur in the case when general block transfer is used). It contains:

- in the case of General_Ciphering, the Invocation_Counter, holding the invocation field of the initialization vector, see 5.3.3.7.3;
- in the case when the APDU has been authenticated, the authentication tag;
- in the case when the APDU has been signed, the digital signature.

6.6 The GET service

Function

The GET service is used with LN referencing. It can be invoked in a confirmed or unconfirmed manner. Its function is to read the value of one or more COSEM interface object attributes. The result can be delivered in a single response or – if it is too long to fit in a single response – in multiple responses, with block transfer.

Semantics

The GET service primitives shall provide parameters as shown in Table 42.

Table 42 – Service parameters of the GET service

	.request	**.indication**	**.response**	**.confirm**
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Request_Type	M	M (=)	–	–
COSEM_Attribute_Descriptor { COSEM_Attribute_Descriptor }	C	C (=)	–	–
COSEM_Class_Id	M	M (=)		
COSEM_Object_Instance_Id	M	M (=)		
COSEM_Object_Attribute_Id	M	M (=)		
Access_Selection_Parameters	U	U (=)		
Access_Selector	M	M (=)		
Access_Parameters	M	M (=)		
Block_Number	C	C (=)	–	–
Response_Type	–	–	M	M (=)
Result	–	–	M	M (=)
Get_Data_Result { Get_Data_Result }	–	–	S	S (=)
Data			S	S (=)
Data_Access_Result			S	S (=)
DataBlock_G	–	–	S	S (=)
Last_Block			M	M (=)
Block_Number			M	M (=)
Result			M	M (=)
Raw_Data			S	S (=)
Data_Access_Result			S	S (=)
NOTE For security parameters see Table 40.				

The Invoke_Id parameter identifies the instance of the service invocation.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The Service_Class parameter indicates whether the service is confirmed or unconfirmed. The handling of this parameter depends on the communication profile; see Annex A.

The use of the Request_Type and Response_Type parameters is shown in Table 43.

Table 43 – GET service request and response types

Request type		Response type	
NORMAL	The value of a single attribute is requested.	NORMAL	The complete result is delivered.
		ONE-BLOCK	One block of the result is delivered.
NEXT	The next data block is requested.	ONE-BLOCK	As above.
		LAST-BLOCK	The last block of the result is delivered.
WITH-LIST	The value of a list of attributes is requested.	WITH-LIST	The complete result is delivered.
		ONE-BLOCK	As above.
NOTE The same Response_Type can be present more than once, to show the possible responses to each kind of request.			

The COSEM_Attribute_Descriptor parameter references a COSEM object attribute. It is present when Request_Type == NORMAL or WITH-LIST. It is a composite parameter:

- the (COSEM_Class_Id, COSEM_Object_Instance_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM_Object_Attribute_Id element identifies the attribute(s) of the object instance. COSEM_Object_Attribute_Id == 0 references all public attributes of the object (Attribute_0 feature; see 4.2.4.3.7);
- the Access_Selection_Parameters is present only when COSEM_Object_Attribute_Id != 0 and if selective access to the given attribute is available; see 4.2.4.3.5. The Access_Selector and Access_Parameters sub-parameters are defined in the COSEM interface object definitions; see IEC 62056-6-2:2021.

A GET-REQUEST-NORMAL service primitive contains a single COSEM attribute descriptor. A GET-REQUEST-WITH-LIST service primitive contains a list of COSEM attribute descriptors; their number is limited by the server-max-receive-pdu-size: a GET.request service primitive shall always fit in a single APDU.

The Block_Number parameter is used in the GET-REQUEST-NEXT service primitive. It carries the number of the latest data block received correctly.

If the encoded form of the response fits in a single APDU, the Result is of type Get_Data_Result:

- the Data choice carries the value of the attribute at the time of access; or
- the Data_Access_Result choice carries the reason for the read to fail for this attribute.

A GET-RESPONSE-NORMAL service primitive carries a single Get_Data_Result parameter. A GET-RESPONSE-WITH-LIST service primitive carries a list of Get_Data_Result parameters; their number and order shall be the same as that of the COSEM_Attribute_Descriptor parameters in the request.

If COSEM_Object_Attribute_Id == 0 (Attribute_0), the Data shall be a structure containing the value of all public attributes in the order of their appearance in the given object specification. For attributes to which no access right is granted within the given AA, or which cannot be accessed for any other reason, null-data shall be returned.

If the encoded form of the response does not fit in a single APDU, it can be transported in data blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used, the Result is of type DataBlock_G. It carries block transfer control information and raw-data:

- the Last_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block_Number element carries the number of the actual block sent;
- the (inner) Result element carries either Raw_Data or Data_Access_Result. Within this:
  - if the value of a single attribute was requested, Raw_Data carries a part of the value of the attribute (Data). If the Data cannot be delivered, the response shall be GET-RESPONSE-NORMAL with Data_Access_Result;
  - if the value of a list of attributes was requested, Raw_Data carries a part of the list of Get_Data_Results, either Data or Data_Access_Result for each attribute;
  - if the raw data cannot be delivered, Data_Access_Result shall carry the reason. For error cases, see 7.3.3.

Use

Possible logical sequences of the GET service primitives are illustrated in Figure 35:

- for a successful confirmed GET, item a);
- for an unconfirmed GET, item d); and
- for an unsuccessful attempt due to a local error, item c).

The GET.request primitive is invoked by the client AP to read the value of one or all attributes of one or more COSEM objects of the server AP. The first request shall always be of Request_Type == NORMAL or WITH-LIST. A GET-REQUEST-NEXT service primitive is invoked only when the server could not deliver the complete data in a single response, i.e. following the reception of a .confirm primitive of Response_Type == ONE-BLOCK. Upon reception of the .request primitive, the client AL builds the Get-Request APDU appropriate for the request type and sends it to the server.

The GET.indication primitive is generated by the server AL upon reception of a Get-Request APDU.

The GET.response primitive is invoked by the server AP, if Service_Class == Confirmed, to send a response to an .indication primitive received. If the complete data requested fits in a single APDU, the .response primitive is invoked with Response_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Response_Type == ONE-BLOCK and finally with LAST-BLOCK.

The GET.confirm primitive is generated by the client AL to indicate the reception of a Get-Response APDU.

The protocol for the GET service is specified in 7.3.3.

6.7 The SET service

Function

The SET service is used with LN referencing. It can be invoked in a confirmed or unconfirmed manner. Its function is to write the value of one or more COSEM interface object attributes. The data to be written can be sent in a single request or – if it is too long to fit in a single request – in multiple requests, with block transfer.

Semantics

The SET service primitives shall provide parameters as shown in Table 44.

Table 44 – Service parameters of the SET service

	.request	**.indication**	**.response**	**.confirm**
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Request_Type	M	M (=)	–	–
COSEM_Attribute_Descriptor { COSEM_Attribute_Descriptor }	C	C (=)	–	–
COSEM_Class_Id	M	M (=)		
COSEM_Object_Instance_Id	M	M (=)		
COSEM_Object_Attribute_Id	M	M (=)		
Access_Selection_Parameters	U	U (=)		
Access_Selector	M	M (=)		
Access_Parameters	M	M (=)		
Data {Data }	C	C (=)	–	–
DataBlock_SA	C	C (=)	–	–
Last_Block	M	M (=)		
Block_Number	M	M (=)		
Raw_Data	M	M (=)		
Response_Type	–	–	M	M (=)
Result { Result }	–	–	C	C (=)
Block_Number	–	–	C	C (=)
NOTE For security parameters, see Table 40.				

The Invoke_Id parameter identifies the instance of the service invocation.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The Service_Class parameter indicates whether the service is confirmed or unconfirmed. The handling of this parameter depends on the communication profile; see Annex A.

The use of the Request_Type and Response_Type parameters is shown in Table 45.

Table 45 – SET service request and response types

Request type		Response type	
NORMAL	The reference of a single attribute and the complete data to be written is sent.	NORMAL	The result is delivered.
FIRST-BLOCK	The reference of a single attribute and the first block of the data to be written is sent.	ACK-BLOCK	The correct reception of the block is acknowledged.
ONE-BLOCK	One block of the data to be written is sent.		
LAST-BLOCK	The last block of the data to be written is sent.	LAST-BLOCK	The correct reception of the last block is acknowledged and the result is delivered.
		LAST-BLOCK-WITH-LIST	The correct reception of the last block is acknowledged and the list of results is delivered.
WITH-LIST	The reference of a list of attributes and the complete data to be written is sent.	WITH-LIST	The list of results is delivered.
FIRST-BLOCK-WITH-LIST	The reference of a list of attributes and the first block of data to be written is sent.	ACK-BLOCK	The correct reception of the block is acknowledged.
NOTE The same Response_Type can be present more than once, to show the possible responses to each request.			

The COSEM_Attribute_Descriptor parameter references a COSEM object attribute. It is present when Request_Type == NORMAL, FIRST-BLOCK, WITH-LIST and FIRST-BLOCK-WITH-LIST. It is a composite parameter:

- the (COSEM_Class_Id, COSEM_Object_Instance_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM_Object_Attribute_Id identifies the attribute(s) of the object instance. COSEM_Object_Attribute_Id == 0 references all public attributes of the object (Attribute_0 feature; see 4.2.4.3.7);
- the Access_Selection_Parameters is present only when COSEM_Object_Attribute_Id != 0 and if selective access to the given attribute is available; see 4.2.4.3.5. The Access_Selector and the Access_Parameters sub-parameters are defined in the COSEM interface object definitions; IIEC 62056-6-2:2021.

A SET-REQUEST-NORMAL or SET-REQUEST-WITH-FIRST-BLOCK service primitive contains a single COSEM attribute descriptor. A SET-REQUEST-WITH-LIST or a SET-REQUEST-FIRST-BLOCK-WITH-LIST service primitive contains a list of COSEM attribute descriptors; their number is limited by the server-max-receive-pdu-size: all COSEM attribute descriptors – together with (a part of) the data to be written – shall fit in a single APDU.

The Data parameter contains the data necessary to write the value of the referenced attributes. The number and the order of the Data parameters shall be the same as that of the COSEM_Attribute_Descriptor parameters.

If COSEM_Object_Attribute_Id == 0 (Attribute_0), the Data sent shall be a structure, containing, for each public attribute, in the order of their appearance in the given object specification, either a value or null-data, meaning that the given attribute need not be set.

If the encoded form of the Data parameter does not fit in a single APDU, it can be transported in blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used, the DataBlock_SA parameter carries block transfer control information and raw-data:

- the Last_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block_Number element carries the number of the actual block sent;
- the Raw_Data element carries a part of the data to be written.

The Result parameters are present in the .response primitive when Response_Type != ACK-BLOCK. Their number and order shall be the same as that of the COSEM_Attribute_Descriptor parameters in the request. Each Result shall contain either the information "success" or a reason for failing to write the attribute referenced (Data_Access_Result). When in the .request primitive COSEM_Object_Attribute_Id == 0 (Attribute_0), the Result shall carry a list of results, either the information "success" or a reason for failing to write the attribute (Data_Access_Result), for each public attribute, in the order of their appearance in the given object specification.

The Block_Number parameter shall be present when Response_Type == ACK-BLOCK, LAST-BLOCK, or LAST-BLOCK-WITH-LIST. It carries the number of the latest data block received correctly.

Use

Possible logical sequence of the SET service primitives are illustrated in Figure 35:

- for a successful confirmed SET, item a);
- for an unconfirmed SET, item d); and
- for an unsuccessful attempt due to a local error, item c).

The SET.request primitive is invoked by the client AP to write the value of one or all attributes of one or more COSEM objects of the server AP. If the complete data to be sent fits in a single APDU, the .request primitive shall be invoked with Request_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it shall be invoked with Request_Type == FIRST-BLOCK or FIRST-BLOCK-WITH-LIST, then with Request_Type == ONE-BLOCK and finally with LAST-BLOCK as appropriate. Upon reception of the .request primitive, the client AL builds the Set-Request APDU appropriate for the Request_Type and sends it to the server.

The SET.indication primitive is generated by the server AL upon reception of a Set-Request APDU.

The SET.response primitive is invoked by the server AP, if Service_Class == Confirmed, to send a response to an .indication primitive received. If the data were sent in a single APDU, the .response primitive is invoked with Response_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Response_Type == ACK-BLOCK, and finally with LAST-BLOCK or LAST-BLOCK-WITH-LIST as appropriate.

The SET.confirm primitive is generated by the client AL to indicate the reception of a Set-Response APDU.

The protocol for the SET service is specified in 7.3.4.

6.8 The ACTION service

Function

The ACTION service is used with LN referencing. It can be invoked in a confirmed or unconfirmed manner. Its function is to invoke one or more COSEM interface objects methods. It comprises two phases:

- in the first phase, the client sends the reference(s) of the method(s) to be invoked, with the method invocation parameters necessary;
- in the second phase, after invoking the methods, the server sends back the result and the return parameters generated by the invocation of the method(s), if any.

If the method invocation parameters are too long to fit in a single request, they are sent in multiple requests (block transfer from the client to the server). If the result and the return parameters are too long to fit in a single response, they are returned in multiple responses (block transfer from the server to the client.)

Semantics

The ACTION service primitives shall provide parameters as shown in Table 46.

Table 46 – Service parameters of the ACTION service

	.request	**.indication**	**.response**	**.confirm**
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Request_Type	M	M (=)	–	–
COSEM_Method_Descriptor { COSEM_Method_Descriptor }	C	C (=)	–	–
COSEM_Class_Id	M	M (=)		
COSEM_Object_Instance_Id	M	M (=)		
COSEM_Object_Method_Id	M	M (=)		
Method_Invocation_Parameters { Method_Invocation_Parameters }	U	U (=)	–	–
Response_Type	-	-	M	M (=)
Action_Response { Action_Response }	–	–	M	M (=)
Result			M	M (=)
Response_Parameters			U	U (=)
Data			S	S (=)
Data_Access_Result			S	S (=)
DataBlock_SA	C	C (=)	C	C (=)
Last_Block	M	M (=)	M	M (=)
Block_Number	M	M (=)	M	M (=)
Raw_Data	M	M (=)	M	M (=)
Block_Number	C	C (=)	C	C (=)
NOTE For security parameters, see Table 40.				

The Invoke_Id parameter identifies the instance of the service invocation.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The Service_Class parameter indicates whether the service is confirmed or unconfirmed. The handling of this parameter depends on the communication profile; see Annex A.

The use of the Request_Type and Response_Type parameters is shown in Table 47.

Table 47 – ACTION service request and response types

Request type		Response type	
NORMAL	The reference of a single method and the complete method invocation parameter is sent.	NORMAL	The result and the complete return parameter are sent.
		ONE-BLOCK	One block of the result and of the return parameter is sent.
NEXT	The next data block is requested.	ONE-BLOCK	As above.
		LAST-BLOCK	The last block of the result(s) and of the return parameter(s) is sent.
FIRST-BLOCK	The reference of a single method and the first block of the method invocation parameters is sent.	NEXT	Correct reception of the block is acknowledged.
ONE-BLOCK	One block of the method invocation parameters is sent.		
LAST-BLOCK	The last block of the method invocation parameters is sent.	NORMAL	As above.
		ONE-BLOCK	As above.
WITH-LIST	The reference of a list of methods and the complete list of method invocation parameters is sent.	WITH-LIST	The complete list of results and return parameters is sent.
		ONE-BLOCK	See above.
WITH-LIST-AND-FIRST-BLOCK	The reference of a list of methods and the first block of the method invocation parameters is sent.	NEXT	The correct reception of the block is acknowledged.
NOTE The same Response_Type can be present more than once, to show the possible responses to each request.			

The COSEM_Method_Descriptor parameter references a COSEM object method. It is present if Request_Type == NORMAL, FIRST-BLOCK, WITH-LIST and WITH-LIST-AND-FIRST-BLOCK. It is a composite parameter:

- the (COSEM_Class_Id, COSEM_Object_Instance_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM_Method_Id identifies one method of the COSEM object referenced.

An ACTION-REQUEST-NORMAL or ACTION-REQUEST-FIRST-BLOCK service primitive shall contain a single COSEM method descriptor. An ACTION-REQUEST-WITH-LIST or ACTION-REQUEST-WITH-LIST-AND-FIRST-BLOCK service primitive shall contain a list of COSEM method descriptors; their number is limited by the server-max-receive-pdu-size: all COSEM method references – together with (a part of) the method invocation parameters – shall fit in a single APDU.

The Method_Invocation_Parameter parameter carries the parameter(s) necessary for the invocation of the method(s) referenced.

- if Request_Type == NORMAL, the Method_Invocation_Parameter parameter is optional;
- if Request_Type == WITH-LIST, the service primitive shall contain a list of Method_Invocation_Parameters. The number and the order of the method invocation parameters shall be the same as that of the COSEM_Method_Descriptor-s. If the invocation of any of the methods does not require additional parameters, it shall be nevertheless present, but it shall be null data.

If the encoded form of the COSEM method descriptor(s) and method invocation parameter(s) does not fit in a single APDU, it can be transported in blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used the DataBlock_SA parameter carries block transfer control information and raw-data:

- the Last_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block_Number element carries the number of the actual block sent;
- the Raw_Data element carries a part of the method invocation parameters.

The Action_Response parameters are present in the .response primitive when Response_Type == NORMAL, or WITH-LIST. Their number and the order shall be the same as that of the of COSEM method descriptors. It consists of two elements:

- the Result parameter. It contains either the information “success” or a reason for failing to invoke the method referenced (Action-Result);
- the Response_Parameter(s). Each response parameter shall contain either the Data returned as a result of invoking the method, or a reason for returning the parameters to fail (Data-Access-Result). If the invocation of any of the methods does not return parameters, null data shall be returned.

If the response does not fit in a single APDU, it can be transported in blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used, the DataBlock_SA parameter carries block transfer control information and raw-data:

- the Last_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block_Number element carries the number of the actual block sent;
- the Raw_Data element carries a part of the response:
  – if a single method was invoked, Raw_Data carries the result and the Response_Parameters if any. If no Response_Parameters are returned, the response shall be of type ACTION-RESPONSE-NORMAL;
  – if a list of methods was invoked, Raw_Data carries a part of the list of Action_Responses and optional data.

The Block_Number parameter in an ACTION-REQUEST-NEXT service primitive shall carry the number of the latest data block received from the server correctly.

The Block_Number parameter in an ACTION-RESPONSE-NEXT service primitive shall carry the number of the latest data block received from the client correctly.

Use

Possible logical sequences of the ACTION service primitives is illustrated in Figure 35:

- for a successful confirmed ACTION, item a);
- for an unconfirmed ACTION, item d); and
- for an unsuccessful attempt due to a local error, item c).

In the first phase, the ACTION.request primitive is invoked by the client AP to invoke one or more methods of one or more COSEM interface objects of the server AP. If the complete list of

COSEM method descriptors and method invocation parameters fits in a single APDU, the .request primitive is invoked with Request_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Request_Type == FIRST-BLOCK or WITH-LIST-AND-FIRST-BLOCK as appropriate, then with Request_Type == ONE-BLOCK and finally with LAST-BLOCK. Upon reception of the .request primitive, the client AL builds the Action-Request APDU appropriate for the Request_Type and sends it to the server.

The ACTION.indication primitive is generated by the server AL upon reception of an Action-Request APDU.

During the block transfer of the method invocation parameters, the server AP invokes the ACTION.response primitive with Request_Type == NEXT until the last block is received.

The ACTION.confirm primitive is generated by the client AL upon reception of an Action-Response APDU.

Once all method invocation parameters are transferred, the server invokes the methods of COSEM interface objects referenced, and the second phase commences.

If the complete response fits in a single APDU, the ACTION.response primitive is invoked by the server AP with Response_Type == NORMAL or WITH-LIST, as appropriate. Otherwise, it is invoked with Response_Type == ONE-BLOCK and finally with LAST-BLOCK. Upon reception of the .response primitive, the server AL builds the Action-Response APDU appropriate for the Response_Type and sends it to the client.

The ACTION.confirm primitive is generated by the client AL to indicate the reception of an Action-Response APDU.

During the block transfer of the return parameters, the client AP invokes the .request primitive with Request_Type == NEXT until the last block is received.

The protocol for the ACTION service is specified in 7.3.5.

6.9 The ACCESS service

6.9.1 Overview – Main features

6.9.1.1 General

The ACCESS service is a unified service which can be used to access multiple COSEM object attributes and/or methods with a single .request / .response. The purpose of introducing it is to improve xDLMS messaging while maintaining co-existence with the existing xDLMS services.

6.9.1.2 Unified WITH-LIST service to improve efficiency

The ACCESS service is a unified service using LN referencing that can be used to read or write multiple COSEM object attributes and/or to invoke multiple methods with a single .request / .response. Each request contains a list of requests and related data. Each response contains a list of return data and the result of the request.

NOTE SN referencing is currently not supported. It can be added by introducing new variants of the service.

Whereas GET- / SET- / ACTION-WITH-LIST service requests can include one request type – GET, SET and/or ACTION – only on the list, ACCESS service requests can include different request types. This allows reducing the number of exchanges and thereby improves efficiency.

The processing of the list of requests starts at the first request on the list and continues with processing the next one until the end is reached.

6.9.1.3 Specific variants for selective access

The GET / SET .request service primitives shall always contain Access_Selection_Parameters even in the case when selective access is not available or not needed. In contrast, the ACCESS service provides specific variants to access attributes without or with selective access. This obviates the need to include Access_Selection_Parameters when selective access is not available or not needed thereby reducing overhead and improving efficiency.

6.9.1.4 Long_Invoke_Id parameter

The Invoke-Id parameter of the GET, SET and ACTION services allows the client and the server to pair requests and responses. The range of the Invoke_Id is 0…15.

In some cases this is not sufficient. To support those cases, the ACCESS service uses a Long_Invoke_Id parameter. The range of the Long_Invoke_Id is 0…16 777 215.

NOTE Description of the circumstances when long Invoke_id-s are useful is beyond the Scope of this International Standard.

6.9.1.5 Self-descriptive responses

When requested by the client, the ACCESS.response service primitive carries not only the response to each request, i.e. the result of accessing each attribute / method and the return data but also the Access_Request_Specification service parameter – carrying the attribute / method references and where applicable, the Access_Selection parameters – rendering the .response service primitive self-descriptive. Such self-descriptive responses can be stored and processed on their own, without the need to pair responses and requests.

6.9.1.6 Failure management

In the case of the GET- / SET- / ACTION-WITH-LIST services the client cannot control what should happen if one of the requests fails. In contrast, the ACCESS service allows the client to control if the requests that follow the failed one on the list should be processed or not.

6.9.1.7 Time stamp as a service parameter

The xDLMS services specified earlier do not provide a service parameter in the .request or in the .response service primitive to carry a time stamp.

In contrast, ACCESS service primitives provide a service parameter to carry the time stamp holding the date and time of invoking the service primitive. This further reduces overhead.

6.9.1.8 Presence of data in service primitives

There are important differences between the GET / SET / ACTION services and the ACCESS service as regards to data in the service primitives:

- GET service: data is not present in the request. In the response, either data or result (Data-Access-Result) is returned;
- SET service: data is present in the request. In the response only result (Data-Access-Result) is returned;
- ACTION service: method invocation parameters are optional in the request. In the response the result of invoking the method (Action-Result) and optionally the result of returning the return parameters (Data or Data-Access-Result) is returned;

- ACCESS service: data is associated with each attribute / method reference in the request. If data is not needed for a particular request, then null-data is included. In the response, both data and result are returned. If there is no data to return for a particular response, then null-data is included. In the case of accessing a method, Access-Response-Action (Action-Result) conveys both the result of invoking the method and the result of returning the return parameters.

6.9.2 Service specification

Function

The ACCESS service is a unified service using LN referencing that can be used to read or write multiple COSEM object attributes and/or to invoke multiple methods with a single .request / .response. Each request contains a list of requests and related data. Each response contains a list of return data and the result of the request. It can be invoked in a confirmed or unconfirmed manner. It can be used with the general block transfer and general ciphering mechanisms.

The use of the conformance block is the following:

- bit 17 *access* indicates the support of the ACCESS service;
- bit 1 *general-protection* indicates the availability of the general protection APDUs;
- bit 2 *general-block-transfer* indicates the availability of the GBT mechanism;
- bit 8 *attribute0-supported-with-set* and bit 10 *attribute0-supported-with-get* (10) are not relevant: attribute0 is always supported;
- bit 9 priority-mgmt-supported is relevant;
- bit 14 *multiple-references* is irrelevant: the ACCESS service always supports multiple references;
- bit 21 *selective-access* is relevant. The access selection parameters can be used only if the use of selective access has been successfully negotiated.

Semantics

The ACCESS service primitives shall provide parameters as shown in Table 48.

The Long_Invoke_Id, Self_Descriptive, Processing_Option, Service_Class and Priority parameters are mandatory. Their value in the .indication, .response and .confirm service primitives shall be the same as in the .request service primitive. They are carried by the bits of the long-invoke-id-and-priority field of the access-request / access-response APDU:

long-invoke-id (bits 0-23) identifies the instance of the service invocation;

- self-descriptive (bit 28) indicates if the service response shall be not self-descriptive (FALSE) or self-descriptive (TRUE). When set to TRUE, the Access_Response_Body parameter shall contain the Access_Request_Specification parameter;

  NOTE 1 The Access_Request_List_Of_Data parameter is not included in the .response service primitive.

- processing-option (bit 29) specifies what to do when processing a request on the list fails. When set to FALSE, processing continues. When set to TRUE, processing breaks i.e. the requests on the list that follow the failed one shall not be processed. As described in 6.9.1.2, processing of the list of requests shall start at the first request on the list and shall continue with processing the next one until the end of the list is reached;
- service-class (bit 30) indicates whether the service invocation is confirmed (TRUE) or unconfirmed (FALSE);

  NOTE 2 The Service_Class parameter applies to the service invocation, not to the individual requests on the list.

NOTE 3 Depending on the communication profile, the Service_Class parameter may also determine the frame type to be used to carry the APDU.

- priority (bit 31) indicates the priority level associated to the instance of the service invocation. It may be normal (FALSE) or high (TRUE).

NOTE 4 The Priority parameter applies to the service invocation, not to the individual requests on the list.

The Date_Time service parameter is optional. When present, it shall contain the date and time of the invocation of the service .request / .response. It is carried by the date-time field – of type OCTET STRING – of the access-request / access-response APDU. When not present, then the OCTET STRING shall be of length 0. Unless otherwise specified in a project specific companion specification, the Date_Time parameter in the response shall be present if it was present in the request and shall not be present in the response if it was not present in the request.

Table 48 – Service parameters of the ACCESS service

	.request	.indication	.response	.confirm
Long_Invoke_Id	M	M (=)	M (=)	M (=)
Self_Descriptive	M	M (=)	M (=)	M (=)
Processing_Option	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Date_Time	U	U (=)	U	U (=)
Access_Request_Body	M	M (=)	–	–
Access_Request_Specification	M	M (=)	–	–
{ Access_Request_Specification }				
Access_Request_Get	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Request_Set	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Request_Action	U	U (=)	–	–
COSEM_Method_Descriptor	M	M (=)	–	–
Access_Request_Get_With_Selection	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Selection	M	M (=)	–	–
Access_Selector	M	M (=)	–	–
Access_Parameters	M	M (=)	–	–
Access_Request_Set_With_Selection	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Selection	M	M (=)	–	–
Access_Selector	M	M (=)	–	–
Access_Parameters	M	M (=)	–	–
Access_Request_List_Of_Data	M	M (=)	–	–
Data { Data }			–	–
Access_Response_Body	–	–	M	M (=)
Access_Request_Specification	–	–	C (=) [1]	C (=)
{ Access_Request_Specification }				
Access_Response_List_Of_Data	–	–	M	M (=)
Data { Data }				
Access_Response_Specification	–	–	M	M (=)
{ Access_Response_Specification}				
Access_Response_Get	–	–	C	C (=)
Result	–	–	M	M (=)
Access_Response_Set	–	–	C	C (=)
Result	–	–	M	M (=)
Access_Response_Action	–	–	C	C (=)
Result	–	–	M	M (=)

[1] When the Access_Request_Specification service parameter is present in Access_Response_Body, then its value shall be the same as in the .request / .indication primitive.

The Access_Request_Body parameter contains the Access_Request_Specification and the Access_Request_List_Of_Data sub-parameters.

The Access_Request_Specification parameter carries a list of request specifications. The list may have 0 or more elements. Each request can be any of the following:

Without selective access:

- Access_Request_Get carries the COSEM_Attribute_Descriptor of an attribute to be read;
- Access_Request_Set carries the COSEM_Attribute_Descriptor of an attribute to be written;
- Access_Request_Action carries the COSEM_Method_Descriptor of a method to be invoked.

With selective access:

- Access_Request_Get_With_Selection carries the COSEM_Attribute_Descriptor of an attribute to be read with selective access, and the Access_Selection parameter that contains Access_Selector and Access_Parameters;
- Access_Request_Set_With_Selection carries the COSEM_Attribute_Descriptor of an attribute to be written with selective access, and the Access_Selection parameter that contains Access_Selector and Access_Parameters.

Access_Request_Get_ / Set_ With_Selection should not be used if selective access to the attribute is not available or if COSEM_Attribute_Descriptor identifies all attributes (Attribute_0).

The COSEM_Attribute_Descriptor parameter is a composite parameter:

- the (COSEM_Class_Id, COSEM_Object_Instance_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM_Object_Attribute_Id element identifies the attribute of the object instance. COSEM_Object_Attribute_Id == 0 references all public attributes of the object.

The Access_Selection parameter carries the Access_Selector and the Access_Parameters sub-parameters. The possible values are defined in the relevant COSEM interface class definitions.

The Access_Request_List_Of_Data parameter carries the list of data related to the list of Access_Request_Specification parameters. The data depend on the kind of access request:

- Access_Request_Get referencing one or all attributes(Attribute_0): null-data;
- Access_Request_Set: the corresponding data carries the value to be written. In the case of referencing all attributes (Attribute_0), the data shall be a structure. The number of elements in the structure shall be the same as the number of attributes specified in the relevant COSEM IC specification. Each element in the structure contains the value to be written or null-data meaning that the given attribute need not be written;
- Access_Request_Action: the corresponding data carries the method invocation parameters, or if not required null-data.

The number and order of the elements on the two lists shall be the same.

The Access_Response_Body parameter contains the following sub-parameters:

- the Access_Request_Specification (optionally, only when the Self_Descriptive == TRUE);
- the Access_Response_List_Of_Data; and
- the Access_Response_Specification.

Notice that in the response Access_Request_List_Of_Data comes first followed by Access_Response_Specification. If data is provided but the result indicates a failure, then the client should discard the data.

The Access_Request_Specification parameter, when present, shall be the same as in the .request / .indication primitives.

The Access_Response_List_Of_Data parameter carries the data resulting from processing the requests. The number of elements on the list shall be the same as on the Access_Request_Specification list. The data depend on the kind of access request:

- Access_Request_Get referencing a single attribute: the value of the attribute requested or null-data when the value of the attribute cannot be returned;
- Access_Request_Get referencing all attributes (Attribute_0): data shall be a structure, containing the value of each attribute. The number of elements in the structure shall be the same as the number of attributes specified in the relevant COSEM IC specification. If the value of an attribute cannot be returned, then null-data shall be included for that attribute. If no attribute values can be returned then a single null-data shall be returned;
- Access_Request_Set referencing one or all attributes (Attribute_0): null-data;
- Access_Request_Action: the return parameters or when not provided, null-data.

The Access_Response_Specification parameter carries the result of each request. The number of elements on this list shall be the same as on the Access_Request_Specification list:

- Access_Request_Get referencing a single attribute: the result of reading the attribute: *success* or reason for the failure;
- Access_Request_Get referencing all attributes (Attribute_0): the result shall be *success* if the value of all attributes to which access right is granted could be returned. Otherwise, it shall be *Data-Access-Result* giving a reason for the failure;
- Access_Request_Set referencing a single attribute: the result of writing the attribute: *success* or a reason for the failure;
- Access_Request_Set referencing all attributes (Attribute_0): the result shall be *success* if the value of all attributes to which access right is granted could be written. Otherwise, it shall be *Data-Access-Result* giving a reason for the failure;
- Access_Response_Action carries the result of invoking a method: *success* or a reason for the failure. The result shall be *success* if the method could be invoked successfully and – when the IC specification specifies return parameters – they could be successfully returned. Otherwise, it shall be *Action-Result* giving a reason for the failure.

If the Processing_Option parameter is set to TRUE and processing a request on the list fails, then for all requests following the failed one, Access_Response_List_Of_Data shall contain null-data and Access_Response_Specification shall carry the reason for the failure.

Use

Possible logical sequences of the ACCESS service primitives are illustrated in Figure 35:

- for a successful confirmed ACCESS, item a);
- for an unconfirmed ACCESS item d); and
- for an unsuccessful attempt due to a local error item c).

The ACCESS.request primitive is invoked by the client AP to read or write the value of a list of COSEM object attributes and/or to invoke a list of methods.

The ACCESS.indication primitive is generated by the server AL upon the reception of an Access-Request APDU.

The ACCESS.response primitive is invoked by the server AP – if Service_Class == Confirmed – to send a response to an .indication primitive received.

The ACCESS.confirm primitive is generated by the client AL to indicate the reception of an access-response APDU.

When the request or the response does not fit in a single APDU, then the general block transfer mechanism can be used. See 4.2.4.4.9.

If the response would be too long to fit in a single APDU but GBT is not supported, the response may be either a list of null-data and a list of results indicating the reason for the failure.

When cryptographic protection is required, the access-request / access-response APDUs can be transported in general-ded-ciphering, general-glo-ciphering, general-ciphering or general-signing APDUs depending on the kind of protection to be applied. See 6.5

The protocol of the ACCESS service is specified in 7.3.6.

6.10 The DataNotification service

Function

The DataNotification service is an unsolicited, unconfirmed or confirmed service. It is used by the server to push data to the client. It is an unconfirmed service. The push process is configured by “Push setup” objects; see IIEC 62056-6-2:2021, 4.4.8.

Semantics

The DataNotification service primitives shall provide parameters as shown in Table 49.

Table 49 – Service parameters of the DataNotification service primitives

	.request	.indication	.response	.confirm
Long_Invoke_Id	M	M (=)	M (=)	M (=)
Self_Descriptive	–	–	–	–
Processing_Option	–	–	–	–
Service_Class	M	M (=)	M (=)	M (=)
Priority	M	M (=)	–	–
Date_Time	U	U (=)	U	U (=)
Notification_Body	M	M (=)	–	–
Result	-	-	-	M

The Long_Invoke_Id parameter identifies the instance of the service invocation.

The Self_Descriptive, Processing_Option and Service_Class parameters are not used in the case of this service.

The Service_Class parameter indicates whether the DataNotification.request service primitive is invoked in a confirmed or unconfirmed manner.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The optional Date_Time parameter indicates the time at which the DataNotification.request / DataNotification.response service primitive is invoked. When the DataNotification.confirm primitive is invoked as a result of a lower layer confirmation, no Date_Time is provided.

The Date_Time parameter indicates the time at which the DataNotification.request service primitive is invoked. It is octet-string, which may be of length zero when transmitting the Date_Time is not required.

The Notification_Body parameter contains the push data.

The Result parameter indicates the confirmation result and is CONFIRMED in case of Service_Class == Confirmed and received Data-Notification-Confirm APDU; or in case of Service_Class == Unconfirmed and received supporting protocol layer confirmation. The Result parameter is SUPPORTING_LAYER_ FAILED in case of supporting protocol layer failure.

Use

A possible logical sequence of the DataNotification service primitives is illustrated in Figure 35 a), b), and d).

The .request primitive is invoked by the server AP to push data to the remote client AP. Upon reception of the .request primitive, the server AL builds the DataNotification APDU.

The .indication primitive is generated by the client AL upon reception of a DataNotification APDU.

The DataNotification.response primitive is invoked by the client AP to confirm the receipt of a confirmable DataNotification. Upon the reception of the .response primitive, the client AL builds the Data-Notification-Confirm APDU and sends it to the server.

The DataNotification.confirm primitive is invoked by the server AL to convey the receipt of a Data-Notification-Confirm APDU or lower layer confirmation to the server AP.

The protocol for the DataNotification service is specified in 7.3.6.

6.11 The EventNotification service

Function

The EventNotification service is an unsolicited, non-client/server type service. It is requested by the server, upon occurrence of an event, in order to inform the client of the value of an attribute, as though it had been requested by the COSEM. It is an unconfirmed service.

Semantics

The EventNotification service primitives shall provide parameters as shown in Table 50.

Table 50 – Service parameters of the EventNotification service primitives

	.request	.indication
Time	U	U (=)
Application_Addresses	U	U (=)
COSEM_Attribute_Descriptor	M	M (=)
COSEM_Class_Id	M	M (=)
COSEM_Object_Instance_Id	M	M (=)
COSEM_Object_Attribute_Id	M	M (=)
Attribute_Value	M	M (=)

The optional Time parameter indicates the time at which the EventNotification.request service primitive was issued.

The Application_Addresses parameter is optional. It is present only when the EventNotification service is invoked outside of an established AA. In this case, it contains all protocol specific parameters required to identify the sender and destination APs.

When the .request primitive does not contain the optional Application_Addresses parameter, default addresses shall be used, those of the server management logical device and the client management AP. Both APs are always present and in any protocol profile, they are bound to known, pre-defined addresses.

The (COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id) triplet references non-ambiguously one and only one attribute of a COSEM interface object instance.

The Attribute_Value parameter carries the value of this attribute. More information about the notified event may be obtained by interrogating this COSEM interface object.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

Use

A possible logical sequence of the EventNotification service primitives is illustrated in Figure 35 f) and g).

The .request primitive is invoked by the server AP to send the value of a COSEM interface object attribute to the remote client AP. Upon reception of the .request primitive, the Server AL builds the EventNotificationRequest APDU.

In some cases, the supporting lower layer protocol(s) do (does) not allow sending a protocol data unit in a real, unsolicited manner. In these cases, the client has to explicitly solicit sending an EventNotification frame, by invoking the Trigger_EventNotification_Sending service primitive.

The EventNotification.indication primitive is generated by the client AL upon reception of an EventNotificationRequest APDU.

The protocol for the EventNotification service is specified in 7.3.8.

6.12 The TriggerEventNotificationSending service

Function

The function of the TriggerEventNotificationSending service is to trigger the server by the client to send the frame carrying the EventNotification.request APDU.

This service is necessary in the case of protocols, when the server is not able to send a real non-solicited EventNotification.request APDU.

Semantics

The TriggerEventNotificationSending.request service primitive shall provide parameters as shown in Table 51.

Table 51 – Service parameters of the TriggerEventNotificationSending.request service primitive

	.request
Protocol_Parameters	M

The Protocol_Parameters parameter contains all lower protocol layer dependent information, which is required for triggering the server to send out an eventually pending frame containing an EventNotification.request APDU. This information includes the protocol identifier, and all the required lower layer parameters.

Use

Upon reception of a TriggerEventNotificationSending.request service invocation from the client AP, the client AL shall invoke the corresponding supporting layer service to send a trigger message to the server.

6.13 Variable access specification

Variable_Access_Specification is a parameter of the xDLMS Read / Write / UnconfirmedWrite InformationReport .request / .indication service primitives. Its variants are shown in Table 52:

- Variable_Name identifies a DLMS® named variable;
- Parameterized_Access provides the capability to transport additional parameters;
- Block_Number_Access transports a block number;
- Read_Data_Block_Access transports block transfer control information and raw data;
- Write_Data_Block_Access transports block transfer control information.

The use of the different variants depends on the service and it is described in the respective SN service specifications.

Table 52 – Variable Access Specification

Variable_Access_Specification	**Read .request**	**Write .request**	**Unconfirmed Write.request**	**Information Report**
Kind_Of_Access	M	M	M	M
Variable_Name	S	S	S	M
Detailed_Access	Not used in DLMS®/COSEM			
Parameterized_Access	S	S	S	–
Variable_Name	M	M	M	
Selector	U	U	U	
Parameter	U	U	U	
Block_Number_Access	S	–	–	–
Block_Number	M			
Read_Data_Block_Access	S	–	–	–
Last_Block	M			
Block_Number	M			
Raw_Data	M			
Write_Data_Block_Access	–	S	–	–
Last_Block		M		
Block_Number		M		

6.14 The Read service

Function

The Read service is used with SN referencing. It is a confirmed service. Its functions are:

- to read the value of one or more COSEM interface object attributes. In this case, the encoded form of the .request service primitive shall fit in a single APDU. The result can be delivered in a single response, or – if it is too long to fit in a single response – in multiple responses, with block transfer;
- to invoke one or more COSEM interface object methods, when return parameters are expected. In this case, if either the .request (including the method references and the method invocation parameters) or the .response service primitive (including the results and return parameters) is too long to fit in a single APDU, then block transfer with multiple requests and/or responses can be used.

The Read service is specified in IEC 61334-4-41:1996, 10.4 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS®/COSEM.

Semantics

The Read service primitives shall provide service parameters as shown in Table 53.

Table 53 – Service parameters of the Read service

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M (=)	–	–
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)		
Variable_Name	M	M (=)		
Selector	U	U (=)		
Parameter	U	U (=)		
Read_Data_Block_Access	S	S (=)		
Last_Block	M	M (=)		
Block_Number	M	M (=)		
Raw_Data	M	M (=)		
Block_Number_Access	S	S (=)		
Block_Number	M	M (=)		
Result (+)			S	S (=)
Read_Result { Read_Result }	–	–	M	M (=)
Data			S	S (=)
Data_Access_Error			S	S (=)
Data_Block_Result			S	S (=)
Last_Block			M	M (=)
Block_Number			M	M (=)
Raw_Data			M	M (=)
Block_Number			S	S (=)
Result (–)	–	–	S	S (=)
Error_Type			M	M (=)
NOTE For security parameters, see Table 40.				

The use of the different variants of the Variable-Access-Specification service parameter of the Read.request service primitive and the different choices of the Read.response primitive are shown in Table 54.

If the encoded form of the response does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

Table 54 – Use of the Variable_Access_Specification variants and the Read.response choices

Read.request Variable_Access_Specification		Read.response CHOICE	
Variable_Name (Variable_Name}	References a list[1] of COSEM object attributes.	Data {Data}	Delivers the value of the attribute(s) referenced.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the read to fail.
		Data_Block_Result	Delivers block transfer control information and one block of raw data.
Parameterized_Access {Parameterized_Access}	References a list[1] of COSEM object attributes to be read selectively.	Data {Data}	As above.
		Data_Access_Error {Data_Access_Error}	
		Data_Block_Result	
	References a list[1] of COSEM object methods, with method invocation parameters.	Data {Data}	Delivers the method invocation return parameters. NOTE 1 If parameters are returned, this implies that the method invocation succeeded.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the method invocation to fail.
		Data_Block_Result	As above.
Read_Data_Block_ Access	Carries block transfer control information and one part of encoded form of the COSEM method references and method invocation parameters.	Block_Number	Carries the number of the latest data block received.
Block_Number_Access	Carries the number of the latest data block received.	Data_Block_Result	As above.

NOTE 2 The same Read.response choice can be present more than once, to show the possible responses to each request.

[1] A list may have one or more elements.

The Read.request service primitive may have one or more Variable_Access_Specification parameters.

- the Variable_Name variant is used to reference a complete COSEM object attribute to be read. The request may include one or more variable names;
- the Parameterized_Access variant is used either:
  - to reference a COSEM object attribute to be read selectively. In this case, the Variable_Name element references the COSEM object attribute, the Selector and the Parameter elements carry the access selector and the access parameters respectively as specified in the attribute specification; or
  - to reference a COSEM object method to be invoked. In this case, the Variable_Name element references the method, the Selector element is zero and the Parameter element carries the method invocation parameters (if any) or null data;
  - the request may include one or more parameterized access parameters;

  NOTE 1 With this, the Read service can transport information in both directions, just like the ACTION service used with LN referencing: method invocation parameters from the client to the server and return parameters from the server to the client.

- the Read_Data_Block_Access variant is used when one or more COSEM object methods are invoked and the encoded form of the request does not fit in a single APDU. The request may include a single Read_Data_Block_Access parameter. It carries block transfer control information and raw data:

  - the Last_Block element indicates if the given block is the last one (TRUE) or not (FALSE);
  - the Block_Number element carries the number of the actual block sent;
  - the Raw_Data element carries a part of the encoded form of the list of Variable_Access_Specification parameters (as it would be used without block transfer) including the method references and the method invocation parameters. Here, only the variants Variable_Name and Parameterized_Access are allowed.

- the Block_Number_Access variant is used when the server uses block transfer to send a long response, to confirm the reception of a data block and to request the next data block. The request may include a single Block_Number_Access parameter. It carries the number of the latest data block received correctly.

The Result (+) parameter indicates that the requested service has succeeded.

Without block transfer, the .response / .confirm service primitives contain one or more Read_Result parameters. Their number and order shall be the same as that of the Variable_Name / Parameterized_Access parameters in the .request / .indication primitives.

If the Read service is used to read attribute(s), then:

- the Data choice is taken to carry the value of the attribute at the time of access;
- the Data_Access_Error is taken to carry the reason for the read to fail for this attribute.

If the Read service is used to invoke method(s), then:

- the Data choice is taken to carry the return parameters (if data are returned, this implies that the method invocation succeeded). If there are no return parameters, Data shall be null data;

  NOTE 2 However, if no return data are expected, the Write service is used to invoke methods.

- the Data_Access_Error choice is taken to carry the reason for the method invocation to fail for this method.

In the case of block transfer, the .response / .confirm primitive contains a single Read_Result parameter. The Data_Block_Result choice is taken to carry one block of the response:

- the Last_Block element indicates whether the given block is the last one (TRUE) or not (FALSE);
- the Block_Number element shall carry the number of the block sent;
- the Raw_Data element contains a part of the encoded form of the list of Read_Results.

If the data block cannot be provided, then the .response primitive shall carry a single Result parameter using the Data_Access_Error choice, carrying an appropriate error message, for example (14) data-block-unavailable.

If the block number in the request is not the one expected, or if the next block cannot be delivered, then the Read.response service primitive shall be returned with a single Read_Result parameter, with the choice Data_Access_Error, carrying an appropriate code, for example (19) data-block-number-invalid.

The Block_Number choice is taken when the Read service is used to invoke one or more methods and the request is sent in several blocks, to confirm the correct reception of a data block and to ask for the next block. It carries the number of the latest block received.

The Result (–) parameter indicates that the service previously requested failed. The Error_Type parameter provides the reason for failure. In this case, the server shall send back a ConfirmedServiceError APDU instead of a ReadResponse APDU.

Use

A possible logical sequence of the Read service primitives is illustrated in Figure 35 item a).

The Read.request primitive is invoked following the invocation of a GET or ACTION .request primitive by the client AP and mapping this to a Read.request primitive by the SN_MAPPER ASE. The client AL builds then the ReadRequest APDU and sends it to the server. For LN / SN service mapping, see 6.19.

The Read.indication primitive is generated by the server AL upon reception of a ReadRequest APDU.

The Read.response primitive is invoked by the server AP in order to send a response to a previously received Read.indication primitive. The server AL builds then the ReadResponse APDU and sends it to the client.

The Read.confirm primitive is generated by the client AL following the reception of a ReadResponse APDU. It is then mapped back to a GET or ACTION .confirm primitive by the SN_MAPPER ASE and the GET or ACTION .confirm primitive is generated.

The protocol of the Read service is specified in 7.3.9.

6.15 The Write service

Function

The Write service is used with SN referencing. It is a confirmed service. Its functions are:

- to write the value of one or more COSEM interface object attributes;
- to invoke one or more COSEM interface object methods when no return parameters are expected.

In both cases, if the encoded form of the .request service primitive does not fit in a single APDU, then it can be sent in several requests with block transfer. The .response service primitive shall always fit in a single APDU.

The Write service is specified in IEC 61334-4-41:1996, 10.5 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS®/COSEM.

Semantics

The Write service primitives shall provide service parameters as shown in Table 55.

Table 55 – Service parameters of the Write service

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)	–	–
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)		
Variable_Name	M	M (=)		
Selector	M	M (=)		
Parameter	M	M (=)		
Write_Data_Block_Access	S	S (=)	–	–
Last_Block	M	M (=)		
Block_Number	M	M (=)		
Data { Data }	M	M (=)	–	–
Result (+)	–	–	S	S (=)
Write_Result { Write_Result }	-	-	S	S (=)
Success			S	S (=)
Data_Access_Error			S	S (=)
Block_Number			S	S (=)
Result (–)			S	S (=)
Error_Type			M	M (=)
NOTE For security parameters, see Table 40.				

The use of the different variants of the Variable-Access-Specification service parameter of the Write.request service primitive and the different choices of the Write.response primitive are shown in Table 56. The use of the Data service parameter is also explained.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using either the service-specific or the general block transfer mechanism.

Table 56 – Use of the Variable_Access_Specification variants and the Write.response choices

Write.request Variable_Access_Specification		Write.response CHOICE	
Variable_Name {Variable_Name}	References a list[1] of COSEM object attributes. The Data service parameter carries the data to be written or the method invocation parameter(s).	Success {Success}	Indicates that the attribute referenced could be successfully written.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the write to fail.
Parameterized_Access {Parameterized_Access}	References a list[1] of COSEM object attributes to be written selectively. The Data service parameter carries the data to be written.	Success {Success}	As above.
		Data_Access_Error {Data_Access_Error}	
Write_Data_Block_Access	Carries block transfer control information. The Data service parameter carries raw-data, including the encoded form of the list[1] of COSEM object attribute or method references, and the list of data to be written or the list of method invocation parameters.	Block_Number	Carries the number of the latest data block received.

NOTE The same Write.response choice can be present more than once, to show the possible responses to each request.

[1] A list may have one or more elements.

The Write.request service primitive may have one or more Variable_Access_Specification parameters:

- the Variable_Name variant is used to reference a complete COSEM object attribute to be written or COSEM object method to be invoked. The request may include one or more variable names;
- the Parameterized_Access variant is used to reference a COSEM object attribute to be written selectively. In this case, the Variable_Name element references the COSEM object attribute, the Selector and the Parameter elements carry the access selector and the access parameters respectively as specified in the attribute specification. The request may include one or more Parameterized_Access parameters;

The Data service parameter carries the value(s) to be written to the attribute(s), or the method invocation parameter(s) of the method(s) to be invoked. The number and the order of the Data parameters shall be the same as that of the Variable_Access_Specification parameters.

If the Write.request service primitive does not fit into a single APDU, block transfer may be used. In this case:

- the Write_Data_Block_Access variant of the Variable_Access_Specification carries block transfer control information:
  - the Last_Block element indicates whether the given block is the last one (TRUE) or not (FALSE);
  - the Block_Number element carries the number of the actual block sent;
- the Data parameter carries one part of the list of the attribute references and the list of data to be written, or one part of the list of method references and the list of method invocation parameters.
- The request includes a single Write_Data_Block_Access and a single Data parameter.

The Result (+) parameter indicates that the service requested has succeeded.

The .response / .confirm service primitives contain a list of Write_Result parameters. Their number and order shall be the same as that of the Variable_Name / Parameterized_Access parameters in the .request / .indication service primitives.

Without block transfer, and with block transfer after receiving the last block:

- when the Write service is used to write attribute(s), each element carries either the success of the write access (Success) or a reason for the write to fail for this variable (Data_Access_Error);
- when the Write service is used to invoke method(s), each element carries either the success of the method invocation access (Success) or a reason for the method invocation to fail for this variable (Data_Access_Error).

The Block_Number choice is used during block transfer to confirm the correct reception of a data block and to ask for the next block. It carries the number of the latest block received.

If the block-number in the request is not the one expected, or if the block could not be received correctly, then the Write.response service primitive shall be returned with a single Write_Result parameter, with the choice Data_Access_Error, carrying an appropriate code, for example (19) data-block-number-invalid.

The Result (–) parameter indicates that the service requested has failed. The Error_Type parameter provides the reason for failure. In this case, the server shall send back a ConfirmedServiceError APDU instead of a WriteResponse APDU.

Use

A possible logical sequence of the Write service primitives is illustrated in Figure 35 item a).

The Write.request primitive is invoked following the invocation of a SET or ACTION .request primitive by the client AP and mapping this to a Write.request primitive by the SN_MAPPER ASE. The client AL builds then the WriteRequest APDU and sends it to the server. For LN / SN service mapping, see 6.19.

The Write.indication primitive is generated by the server AL upon reception of a WriteRequest APDU.

The Write.response primitive is invoked by the server AP in order to send a response to a previously received Write.indication primitive. The server AL builds then the WriteResponse APDU and sends it to the client.

The Write.confirm primitive is generated by the client AL following the reception of a WriteResponse APDU. It is mapped then back to a SET or ACTION .confirm primitive by the SN_MAPPER ASE and the SET or ACTION .confirm primitive is generated.

The protocol of the Write service is specified in 7.3.10.

6.16 The UnconfirmedWrite service

Function

The UnconfirmedWrite service is used with SN referencing. It is an unconfirmed service. Its functions are:

- to write the value of one or more COSEM object attributes;
- to invoke one or more COSEM interface object method when no return parameters are expected.

The UnconfirmedWrite.request service primitive shall always fit in a single APDU.

The UnconfirmedWrite service is specified in IEC 61334-4-41:1996, 10.6 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS®/COSEM.

Semantics

The UnconfirmedWrite service primitives shall provide service parameters as shown in Table 57.

Table 57 – Service parameters of the UnconfirmedWrite service

	.request	.indication
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)
Variable_Name	S	S (=)
Parameterized_Access	S	S (=)
Variable_Name	M	M (=)
Selector	M	M (=)
Parameter	M	M (=)
Data { Data }	M	M (=)
NOTE For security parameters see Table 40.		

The use of the different variants of the Variable-Access-Specification service parameter of the UnconfirmedWrite.request service primitive is shown in Table 58. The use of the Data service parameter is also explained.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

Table 58 – Use of the Variable_Access_Specification variants

UnconfirmedWrite.request Variable_Access_Specification	
Variable_Name {Variable_Name}	References a COSEM object attribute. The Data service parameter carries the data to be written or the method invocation parameter(s).
Parameterized_Access {Parameterized_Access}	References a COSEM object attribute with selective access. The Data service parameter carries the data to be written.

The UnconfirmedWrite.request service primitive may have one or more Variable_Access_Specification parameters.

- the Variable_Name variant is used to reference a complete COSEM object attribute to be written or COSEM object method to be invoked;
- the Parameterized_Access variant is used to reference a COSEM object attribute to be written selectively. In this case, the Variable_Name element references the COSEM object

attribute, the Selector and the Parameter elements carry the access selector and the access parameters respectively as specified in the attribute specification.

The Data service parameter carries the value(s) to be written to the attribute(s), or the method invocation parameter(s) of the method(s) to be invoked. The number and the order of the Data parameters shall be the same as that of the Variable_Access_Specification parameters.

Use

A possible logical sequence of the Write service primitives is illustrated in Figure 35 item d).

The UnconfirmedWrite.request primitive is invoked following the invocation of a SET or ACTION .request primitive with Service_Class == Unconfirmed by the client AP and mapping this to an UnconfirmedWrite.request primitive by the SN_MAPPER ASE. The client AL builds then the UnconfirmedWriteRequest APDU and sends it to the server.

The UnconfirmedWrite.indication primitive is generated by the server AL upon reception of a WriteRequest APDU.

The protocol of the UnconfirmedWrite service is specified in 7.3.11.

6.17 The InformationReport service

Function

The InformationReport service is an unsolicited, non-client/server type service. It is requested by a server using SN referencing, upon occurrence of an event, in order to inform the client of the value of one or more DLMS® named variables – mapped to COSEM interface object attributes – as though they had been requested by the client. It is an unconfirmed service.

The InformationReport service is specified in IEC 61334-4-41:1996, 10.7 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification of the InformationReport service is reproduced here, together with the extensions made for DLMS®/COSEM.

Semantics

The InformationReport service primitives shall provide parameters as shown in Table 59.

Table 59 – Service parameters of the InformationReport service

	.request	**.indication**
Current_Time	M	M (=)
Variable_Access_Specification { Variable_Access_Specification }	M	M (=)
Variable_Name	M	M (=)
Data { Data }	M	M(=)

The Current_Time parameter indicates the time at which the InformationReport.request service primitive was issued.

The Variable_Access_Specification parameter of choice Variable_Name specifies one or more DLMS® named variables – mapped to COSEM interface object attributes – the value of which is sent by the server.

The Data parameter carries the value of the DLMS® named variable(s), in the same order as the order of the Variable_Access_Specification parameter(s).

The protocol for the InformationReport service is specified in 7.3.12.

6.18 Client side layer management services: the SetMapperTable.request

Function

The function of the SetMapperTable service is to manage the client SN_MAPPER ASE.

Semantics

There is only one primitive, the .request primitive. It shall provide parameters as follows, see Table 60.

Table 60 – Service parameters of the SetMapperTable.request service primitives

	.request
Mapping_Table	M

The Mapping_table parameter is mandatory. It contains the contents of the attribute *object_list* for the requested server and AA. The structure of the content is defined in IIEC 62056-6-2:2021.

Use

The SetMapperTable.request service is invoked by the client AP to provide mapping information to the client SN_MAPPER ASE. This service does not cause any data transmission between the client and the server. The client AP uses this service primitive, in order to enhance the efficiency of the mapping process if SN referencing is used.

6.19 Summary of services and LN/SN data transfer service mapping

Table 61 and Table 62 provide a summary of the DLMS®/COSEM application layer services.

Table 61 – Summary of ACSE services

Client side	Server side
COSEM-OPEN.request	COSEM-OPEN.indication
COSEM-OPEN.confirm	COSEM-OPEN.response
COSEM-RELEASE.request	COSEM-RELEASE.indication
COSEM-RELEASE.confirm	COSEM-RELEASE.response
COSEM-ABORT.indication	COSEM-ABORT.indication

Table 62 – Summary of xDLMS services

Client side	Server side
LN referencing	
GET.request	GET.indication
GET.confirm	GET.response
SET.request	SET.indication
SET.confirm	SET.response
ACTION.request	ACTION.indication
ACTION.confirm	ACTION.response
ACCESS.request	ACCESS.indication
ACCESS.confirm	ACCESS.response
EventNotification.indication	EventNotification.request
TriggerEventNotificationSending.request	–
DataNotification.indication	DataNotification.request
DataNotification.response	DataNotification.confirm
SN referencing	
Read.request	Read.indication
Read.confirm	Read.response
Write.request	Write.indication
Write.confirm	Write.response
UnconfirmedWrite.request	UnconfirmedWrite.indication
InformationReport.indication	InformationReport.request
DataNotification.indication	DataNotification.request
DataNotification.response	DataNotification.confirm
NOTE The DataNotification service can be can be used in application contexts using either SN referencing or LN referencing.	

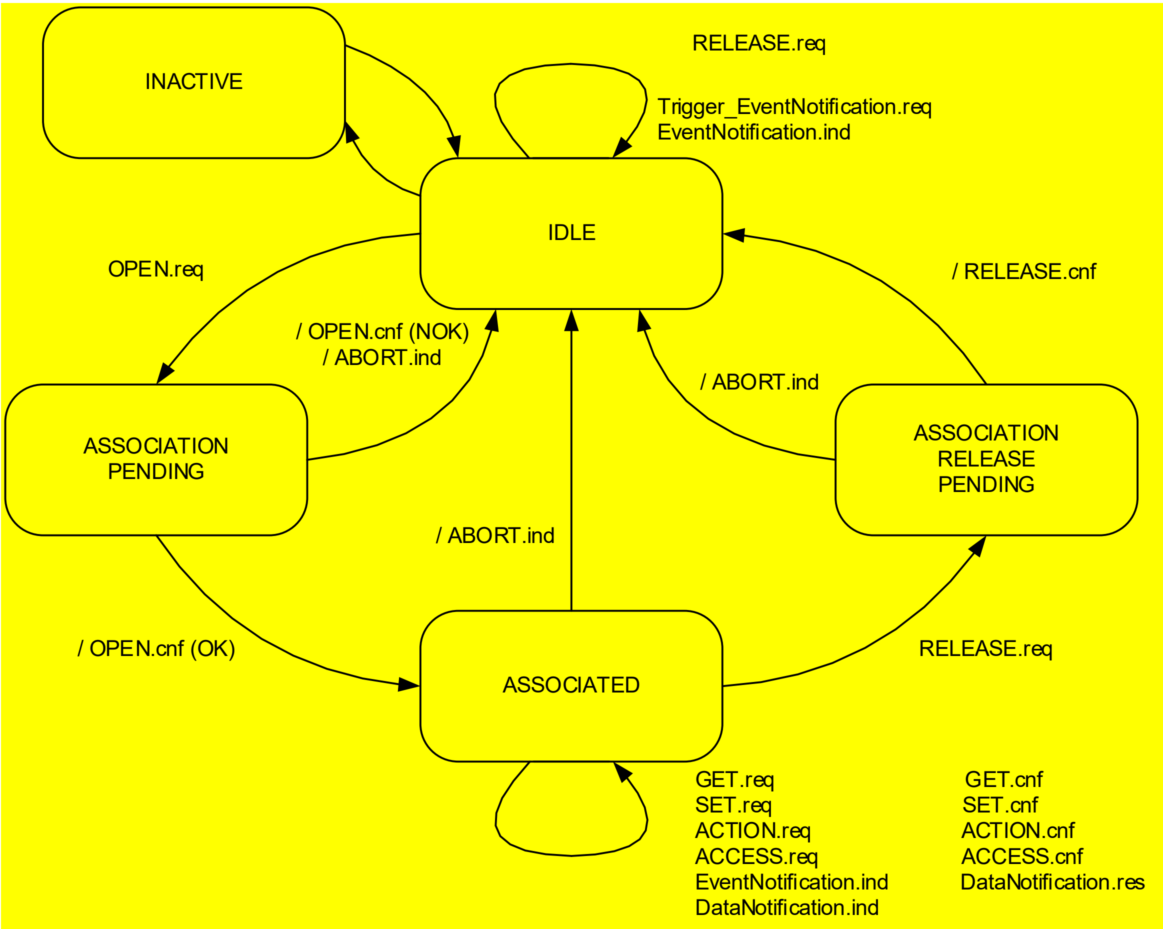
When the server uses SN referencing, a mapping between the service primitives using LN referencing and SN referencing takes place on the client side. This mapping is specified in 7.3.9, 7.3.10, 7.3.11 and 7.3.12.

7 DLMS®/COSEM application layer protocol specification

7.1 The control function

7.1.1 State definitions of the client side control function

Figure 37 shows the state machine for the client side CF, see also Figure 9.



NOTE On the state diagrams of the client and server CF, the following conventions are used:

– service primitives with no "/" character as first character are "stimulants": the invocation of these primitives is the origin of the state transition;

– service primitives with an "/" character as first character are "outputs": the generation of these primitives is done on the state transition path.

Figure 37 – Partial state machine for the client side control function

The state definitions of the client CF – and of the AL including the CF – are as follows:

INACTIVE	In this state, the CF has no activity at all: it neither provides services to the AP nor uses services of the supporting protocol layer.
IDLE	This is the state of the CF when there is no AA existing, being released, or being established [3]. Nevertheless, some data exchange between the client and server – if the physical channel is already established – is possible. The CF can handle the EventNotification service.

[3] Note that it is the state machine for the AL: lower layer connections, including the physical connection, are not taken into account. On the other hand, physical connection establishment is done outside of the protocol.

NOTE State transitions between the INACTIVE and IDLE states are controlled outside of the protocol. For example, it can be considered that the CF makes the state transition from INACTIVE to IDLE by being instantiated and bound on the top of the supporting protocol layer. The opposite transition happens by deleting the given instance of the CF.

ASSOCIATION PENDING	The CF leaves the IDLE state and enters this state when the AP requests the establishment of an AA by invoking the COSEM-OPEN.request primitive (OPEN.req). The CF may exit this state and enter either the ASSOCIATED state or return to the IDLE state, and generates the COSEM-OPEN.confirm primitive, (/OPEN.cnf(OK)) or (/OPEN.cnf(NOK)), depending on the result of the association request. The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATED	The CF enters this state when the AA has been successfully established. All xDLMS services and APDUs are available in this state. The CF remains in this state until the AP requests the release of the AA by invoking the COSEM-RELEASE.request primitive (RELEASE.req). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATION RELEASE PENDING	The CF leaves the ASSOCIATED state and enters this state when the AP requests the release of the AA by invoking the COSEM-RELEASE.request primitive (RELEASE.req). The CF remains in this state, waiting for the response to this request from the server. As the server is not allowed to refuse a release request, after exiting this state, the CF always enters the IDLE state. The CF may exit this state by generating the COSEM-RELEASE.confirm primitive following the reception of a response from the server or by generating it locally (/RELEASE.cnf). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).

7.1.2 State definitions of the server side control function

Figure 38 shows the state machine for the server side CF, see Figure 9.

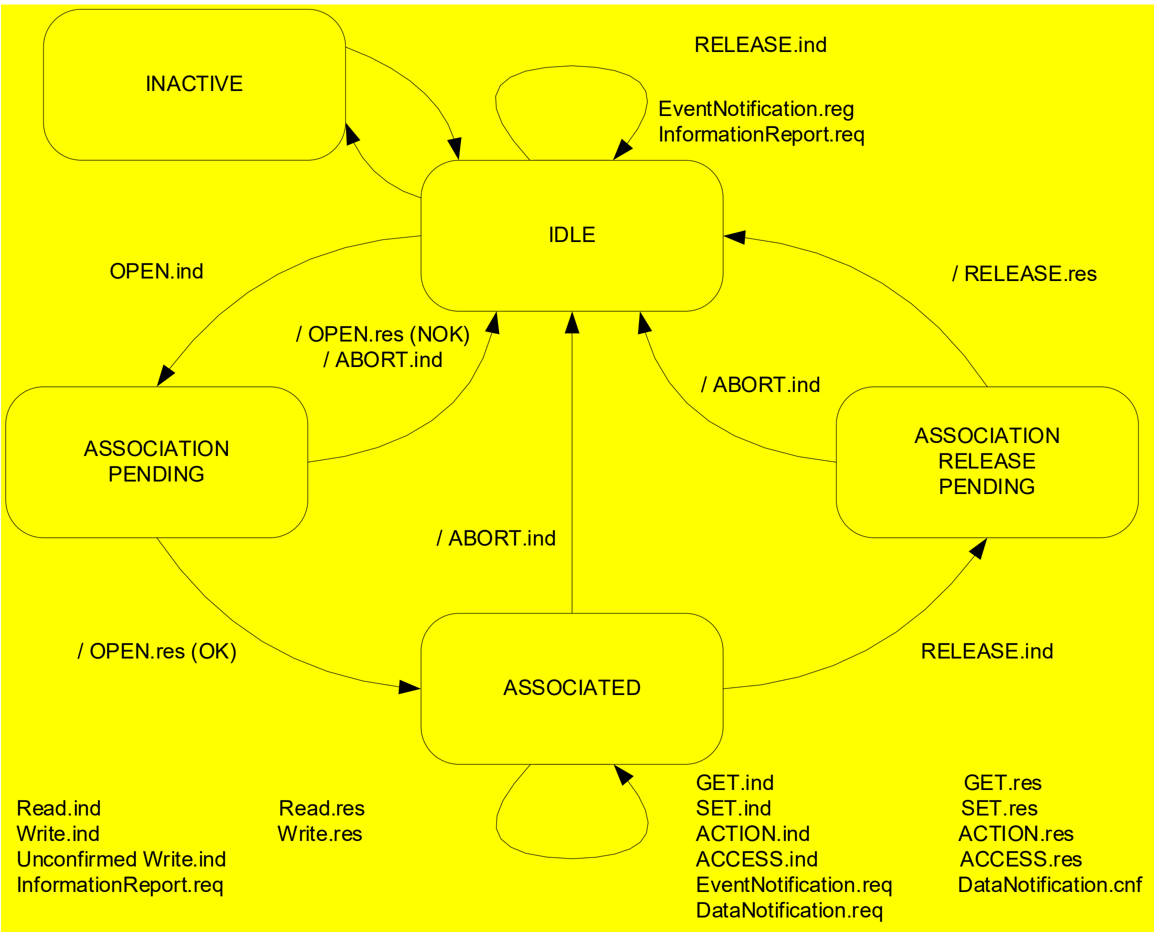


Figure 38 – Partial state machine for the server side control function

INACTIVE	In this state, the CF has no activity at all: it neither provides services to the AP nor uses services of the supporting protocol layer.
IDLE	This is the state of the CF when there is no AA existing, being released, or being established [4]. Nevertheless, some data exchange between the client and server – if the physical channel is already established – is possible. The CF can handle the EventNotification / InformationReport services.
ASSOCIATION PENDING	The CF leaves the IDLE state and enters this state when the client requests the establishment of an AA, and the server AL generates the COSEM-OPEN.indication primitive (/OPEN.ind). The CF may exit this state and enter either the ASSOCIATED state or return to the IDLE state, depending on the result of the association request, and invokes the COSEM-OPEN.response primitive, (/OPEN.res(OK)) or (/OPEN.res(NOK)). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATED	The CF enters this state when the AA has been successfully established. All xDLMS services and APDUs are available in this state. The CF remains in this state until the client requests the release of the AA, and the server AL generates the COSEM-RELEASE.ind primitive (/RELEASE.ind). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATION RELEASE PENDING	The CF leaves the ASSOCIATED state and enters this state when the client requests the release of an AA, and the server AP receives the COSEM-RELEASE.indication primitive (/RELEASE.ind). The CF remains in this state, waiting that the AP accepts the release request. As the server is not allowed to refuse a release request, after exiting this state, the CF always enters the IDLE state. The CF may exit this state when the AP accepts the release of the AA, and invokes the COSEM-RELEASE.response primitive (RELEASE.res). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).

7.2 The ACSE services and APDUs

7.2.1 ACSE functional units, services and service parameters

The DLMS®/COSEM AL ACSE is based on the connection-oriented ACSE, as specified in ISO/IEC 15953 and ISO/IEC 15954.

Functional units are used to negotiate ACSE user requirements during association establishment. Five functional units are defined:

- Kernel functional unit;
- Authentication functional unit;
- ASO-context negotiation functional unit;

  NOTE 1 ISO/IEC 15953:1999 and ISO/IEC 15954:1999 use the term 'ASO-context". In DLMS®/COSEM the term 'Application context" is used as in ISO/IEC 8649 / ISO/IEC 8650-1.
- Higher Level Association functional unit; and
- Nested Association functional unit.

The DLMS®/COSEM AL uses only the Kernel and the Authentication functional unit.

The acse-requirements parameters of the AARQ and AARE APDUs are used to select the functional units for the association.

The Kernel functional unit is always available and includes the basic services A-ASSOCIATE, A-RELEASE.

The Authentication functional unit supports authentication during association establishment. The availability of this functional unit is negotiated during association establishment. This functional unit does not include additional services. It adds parameters to the A-ASSOCIATE service.

[4] Note that it is the state machine for the AL: lower layer connections, including the physical connection, are not taken into account. On the other hand, physical connection establishment is done outside of the protocol.

4914 **Table 63 – Functional Unit APDUs and their fields**

Functional unit	Service	APDU	Field name	Presence
Kernel	A-ASSOCIATE	AARQ	protocol-version	O
			application-context-name	M
			called-AP-title	U
			called-AE-qualifier	U
			called-AP-invocation-identifier	U
			called-AE-invocation-identifier	U
			calling-AP-title	U
			calling-AE-qualifier	U
			calling-AP-invocation-identifier	U
			calling-AE-invocation-identifier	U
			implementation-information	O
			user-information [2)]	M
			(carrying an xDLMS Initiate.request APDU)	
			dedicated-key	U
			response-allowed	U
			proposed-quality-of-service	U
			proposed-dlms-version-number	M
			proposed-conformance	M
			client-max-receive-pdu-size	M
		AARE	protocol-version	O
			application-context-name	M
			result	M
			result-source-diagnostic	M
			responding-AP-title	U
			responding-AE-qualifier	U
			responding-AP-invocation-identifier	U
			responding-AE-invocation-identifier	U
			implementation-information	O
			user-information [3)]	M
			(carrying an xDLMS initiateResponse APDU)	S
			negotiated-quality-of-service	U
			negotiated-dlms-version-number	M
			negotiated-conformance	M
			server-max-receive-pdu-size	M
			vaa-name	M
			(or carrying a confirmedServiceError APDU)	S
	A-RELEASE	RLRQ	reason	U
			user-information	U
		RLRE	reason	U
			user-information	U
Authentication	A-ASSOCIATE	AARQ	sender-acse-requirements	U
			mechanism-name	U
			calling-authentication-value	U

Functional unit	Service	APDU	Field name	Presence
		AARE	responder-acse-requirements	U
			mechanism-name	U
			responding-authentication-value	U
NOTE 1 This table is based on ISO/IEC 15954:1999, Table 2 and 3. The fields are listed in the order as they are in the ACSE APDUs.				
M Presence is mandatory O Presence is ACPM option U Presence is ACSE service-user option S The parameter is selected among other S-parameters as internal response of the server ASE environment.				
NOTE 2 According to ISO/IEC 15953:1999 the user-information parameter is optional. However, in the DLMS®/COSEM environment it is mandatory in the AARQ / AARE APDUs.				
There are several changes in ISO/IEC 15953:1999 and ISO/IEC 15954:1999 compared to ISO/IEC 8649 / ISO/IEC 8650-1: - In ISO/IEC 15954, protocol-version is mandatory in the AARQ and optional in the AARE. In DLMS®/COSEM it is kept as mandatory for backward compatibility; - Instead of “application-context-name”, “ASO-context-name” is used. In DLMS®/COSEM, “application-context-name” is kept. ISO/IEC 15954 7.1.5.2 specifies this: the ASO-context-name is optional. If backward compatibility with older implementations of ACSE is desired, it must be present. Therefore, in DLMS®/COSEM it is mandatory; - In ISO/IEC 15954, the result and result-source-diagnostic parameters are optional. ISO/IEC 15954 7.1.5.8 and 7.1.5.9 specifies this: The Result / Result-source-diagnostic are optional. If backward compatibility with older implementations of ACSE is desired, it must be present. Therefore, in DLMS®/COSEM these parameters are mandatory.				

Table 63 shows the services, APDUs and APDU fields associated with the ACSE functional units, as used by the DLMS®/COSEM AL. The abstract syntax of the ACSE APDUs is specified in Clause 8.

In general, the value of each field of the AARQ APDU is determined by the parameters of the COSEM-OPEN.request service primitive. Similarly, the value if each field of the AARE is determined by the COSEM-OPEN.response primitive. The COSEM-OPEN service is specified in 6.2.

The fields of the AARQ and AARE APDU are specified below. Managing these fields is specified in 7.2.4.1.

- protocol-version: the DLMS®/COSEM AL uses the default value version 1. For details see ISO/IEC 15954:1999;
- application-context-name**:** COSEM application context names are specified in 7.2.2.2;

  NOTE 2 ISO/IEC 15953:1999 and ISO/IEC 15954:1999 use “ASO-context-name”.
- called-, calling- and responding- titles, qualifiers and invocation-identifiers: these optional fields carry the value of the respective parameters of the COSEM-OPEN service. For details see ISO/IEC 15954:1999;
- implementation-information**:** this field is not used by the DLMS®/COSEM AL. For details see ISO/IEC 15954:1999;
- user-information**:** in the AARQ APDU, it carries an xDLMS InitiateRequest APDU holding the elements of the Proposed_xDLMS_Context parameter of the COSEM-OPEN.request service primitive. In the AARE APDU, it carries an xDLMS InitiateResponse APDU, holding the elements of the Negotiated_xDLMS_Context parameter, or an xDLMS confirmedServiceError APDU, holding the elements of the xDLMS_Initiate_Error parameter of the COSEM-OPEN.response service primitive;

- sender- and responder-acse-requirements: this field is used to select the optional functional units of the AARQ / AARE. In COSEM, only the Authentication functional unit is used. When present, it carries the value of BIT STRING { authentication (0) }. Bit set: authentication functional unit selected;
- mechanism-name: COSEM authentication mechanism names are specified in 7.2.2.3;
- calling- and responding- authentication-value: see 5.2.2.2;
- result: the value of this field is determined by the COSEM AP (acceptor) or the DLMS®/COSEM AL (ACPM) as specified below. It is used to determine the value of the Result parameter of the COSEM-OPEN.confirm primitive:
  - if the AARQ APDU is rejected by the ACPM (i.e. the COSEM-OPEN.indication primitive is not issued by the DLMS®/COSEM AL), the value “rejected (permanent)” or “rejected (transient)” is assigned by the ACPM;
  - otherwise, the value is determined by the Result parameter of the COSEM-OPEN.response APDU;
- result-source-diagnostic: this field contains both the Result source value and the Diagnostic value. It is used to determine the value of the Failure_Type parameter of the COSEM-OPEN.confirm primitive:
  - Result-source value: if the AARQ is rejected by the ACPM, (i.e. the COSEM-OPEN.indication primitive is not issued by the DLMS®/COSEM AL) the ACPM assigns the value “ACSE service-provider”. Otherwise, the ACPM assigns the value “ACSE service-user”;
  - Diagnostic value: If the AARQ is rejected by the ACPM, the appropriate value is assigned by the ACPM. Otherwise, the value is determined by the Failure_Type parameter of the COSEM-OPEN.response primitive. If the Diagnostic parameter is not included in the .response primitive, the ACPM assigns the value “null”.

The parameters of the RLRQ / RLRE APDUs – used when the COSEM-RELEASE service (see 6.3) is invoked with the parameter Use_RLRQ_RLRE == TRUE – are specified below.

- reason: carries the appropriate value as specified in 6.2;
- user-information: if present, it carries an xDLMS InitiateRequest / InitiateResponse APDU, holding the elements of the Proposed_xDLMS_Context / Negotiated_xDLMS_Context parameter of the COSEM-RELEASE.request / .response service primitive respectively. See 6.2.

7.2.2 Registered COSEM names

7.2.2.1 General

Within an OSI environment, many different types of network objects must be identified with globally unambiguous names. These network objects include abstract syntaxes, transfer syntaxes, application contexts, authentication mechanism names, etc. Names for these objects in most cases are assigned by the committee developing the particular basic ISO standard or by implementers’ workshops, and should be registered. For DLMS®/COSEM, these object names are assigned by the DLMS® UA, and are specified below.

The decision no. 1999.01846 of OFCOM, Switzerland, attributes the following prefix for object identifiers specified by the DLMS® User Association.

{ joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS®-UA(8) }
NOTE As specified in ITU-T X.660 A.2.4, for historical reasons, the secondary identifiers ccitt and joint-iso-ccitt are synonyms for itu-t and joint-iso-itu-t, respectively, and thus may appear in ASN.1 OBJECT IDENTIFIER values, and also identify the corresponding primary integer value.

For DLMS®/COSEM, object identifiers are specified for naming the following items:

- COSEM application context names;
- COSEM authentication mechanism names;
- cryptographic algorithm ID-s.

7.2.2.2 The COSEM application context

In order to effectively exchange information within an AA, the pair of AE-invocations shall be mutually aware of, and follow a common set of rules that govern the exchange. This common set of rules is called the application context of the AA. The application context that applies to an AA is determined during its establishment.

An AA has only one application context. However, the set of rules that make up the application context of an AA may contain rules for alteration of that set of rules during the lifetime of the AA.

The following methods may be used:

- identifying a pre-existing application context definition;
- transferring an actual description of the application context.

In the COSEM environment, it is intended that an application context pre-exists and it is referenced by its name during the establishment of an AA. The application context name is specified as OBJECT IDENTIFIER ASN.1 type. COSEM identifies the application context name by the following object identifier value:

```
COSEM_Application_Context_Name::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS®-UA(8) application-context(1) context_id(x)}
```

The meaning of this general COSEM application context is:

- there are two ASEs present within the AE invocation, the ACSE and the xDLMS ASE;
- the xDLMS ASE is as it is specified in IEC 61334-4-41:1996

NOTE With the COSEM extensions to DLMS®, see 4.2.4.

- the transfer syntax is A-XDR.

The specific context_id-s and the use of ciphered and unciphered APDUs are shown in Table 64:

Table 64 – COSEM application context names

Application context name	context_id	Unciphered APDUs	Ciphered APDUs
Logical_Name_Referencing_No_Ciphering::=	context_id(1)	Yes	No
Short_Name_Referencing_No_Ciphering::=	context_id(2)	Yes	No
Logical_Name_Referencing_With_Ciphering::=	context_id(3)	Yes	Yes
Short_Name_Referencing_With_Ciphering::=	context_id(4)	Yes	Yes

In order to successfully establish an AA, the application-context-name parameter of the AARQ and AARE APDUs should carry one of the “valid” names. The client proposes an application context name using the Application_Context_Name parameter of the COSEM-OPEN.request primitive. The server may return any value; either the value proposed or the value it supports.

7.2.2.3 The COSEM authentication mechanism name

Authentication of the client, the server or both is one of the security aspects addressed by the DLMS®/COSEM specification. Three authentication security levels are specified:

- no security (Lowest Level Security) authentication, see 5.2.2.2.2;
- Low Level Security (LLS) authentication, see 5.2.2.2.3;
- High Level Security (HLS) authentication, see 5.2.2.2.4.

DLMS®/COSEM identifies the authentication mechanisms by the following general object identifier value:

COSEM_Authentication_Mechanism_Name::= {joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS®-UA(8) authentication_mechanism_name(2) mechanism_id(x)}

The value of the mechanism_id element selects one of the security mechanisms specified:

Table 65 – COSEM authentication mechanism names

COSEM_lowest_level_security_mechanism_name::=	mechanism_id(0)
COSEM_low_level_security_mechanism_name::=	mechanism_id(1)
COSEM_high_level_security_mechanism_name::=	mechanism_id(2)
COSEM_high_level_security_mechanism_name_using_MD5::=	mechanism_id(3)
COSEM_high_level_security_mechanism_name_using_SHA-1::=	mechanism_id(4)
COSEM_high_level_security_mechanism_name_using_GMAC::=	mechanism_id(5)
COSEM_high_level_security_mechanism_name_using_SHA-256::=	mechanism_id(6)
COSEM_high_level_security_mechanism_name_using_ECDSA::=	mechanism_id(7)
NOTE 1 With mechanism_id(2), the method of processing the challenge is secret. NOTE 2 The use of authentication mechanisms 3 and 4 are not recommended for new implementations.	

When the Authentication_Mechanism_Name is present in the COSEM-OPEN service, the authentication functional unit of the A-ASSOCIATE service shall be selected. The process of LLS and HLS authentication is described in 5.2.2.2 and in 5.7.3.

7.2.2.4 Cryptographic algorithm ID-s

Cryptographic algorithm IDs identify the algorithm for which a derived secret symmetrical key will be used. See 5.3.4.6.5.

Cryptographic algorithms are identified by the following general object identifier value:

COSEM_Cryptographic_Algorithm_Id::= {joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS®-UA(8) cryptographic-algorithms (3) algorithm_id(x)}

The values of the algorithm_id-s are shown in Table 66. See also Table 8.

Table 66 – Cryptographic algorithm ID-s

COSEM_cryptographic_algorithm_name_aes-gcm-128::=	algorithm_id(0)
COSEM_cryptographic_algorithm_name_aes-gcm-256::=	algorithm_id(1)
COSEM_cryptographic_algorithm_name_aes-wrap-128::=	algorithm_id(2)
COSEM_cryptographic_algorithm_name_aes-wrap-256::=	algorithm_id(3)

7.2.3 APDU encoding rules

7.2.3.1 Encoding of the ACSE APDUs

The ACSE APDUs shall be encoded in BER (ISO/IEC 8825-1). The user-information parameter of these APDUs shall carry the xDLMS InitiateRequest / InitiateResponse / confirmedServiceError APDU as appropriate, encoded in A-XDR, and then encoding the resulting OCTET STRING in BER.

Examples for AARQ/AARE APDU encoding are given in Annex D

7.2.3.2 Encoding of the xDLMS APDUs

The xDLMS APDUs shall be encoded in A-XDR, as specified in IEC 61334-6.

7.2.3.3 XML

Depending on the parametrization of the “Push setup” object the DataNotification APDU can be encoded as an XML document using the XML schema specified in 9.

NOTE The use of XML to encode the other APDUs is not in the Scope of this International Standard.

7.2.4 Protocol for application association establishment

7.2.4.1 Protocol for the establishment of confirmed application associations

AA establishment using the A-Associate service of the ACSE is the key element of COSEM interoperability. The participants of an AA are:

- a client AP, proposing AAs; and
- a server AP, accepting them or not.

NOTE 1 To support multicast and broadcast services, an AA can also be established between a client AP and a group of server APs.

Figure 39 gives the MSC for the case, when:

- the COSEM-OPEN.request primitive requests a confirmed AA;
- the connection of the supporting lower layers is required for the establishment of this AA.

A client AP that desires to establish a confirmed AA, invokes the COSEM-OPEN.request primitive of the ASO with Service_Class == Confirmed. Note, that the PH layer has to be connected before the COSEM-OPEN service is invoked. The response-allowed parameter of the xDLMS InitiateRequest APDU is set to TRUE. The client AL waits for an AARE APDU, prior to generating the .confirm primitive, with a positive – or negative – result.

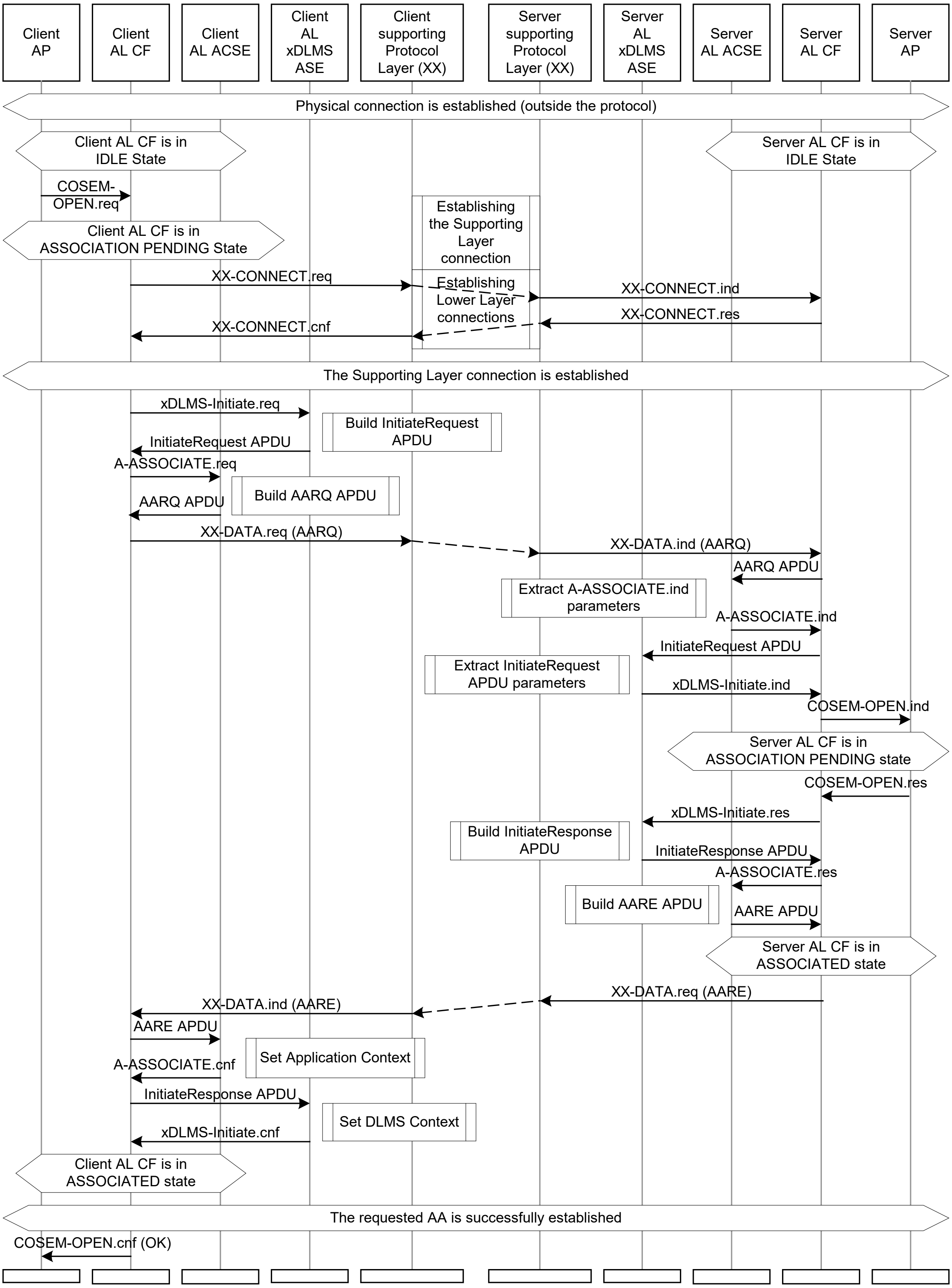
The client CF enters the ASSOCIATION PENDING state. It examines then the Protocol_Connection_Parameters parameter. If this indicates that the establishment of the supporting layer connection is required, it establishes the connection. The CF assembles then

– with the help of the xDLMS ASE and the ACSE – the AARQ APDU containing the parameters of the COSEM-OPEN.request primitive received from the AP and sends it to the server.

The CF of the server AL gives the AARQ APDU received to the ACSE. It extracts the ACSE related parameters then gives back the control to the CF. The CF passes then the contents of the user-information parameter of the AARQ APDU – carrying an xDLMS InitiateRequest APDU – to the xDLMS_ASE. It retrieves the parameters of this APDU, then gives back the control to the CF. The CF generates the COSEM-OPEN.indication to the server AP with the parameters of the APDU received and enters the ASSOCIATION PENDING state.

NOTE 2 Some service parameters of the COSEM-OPEN.indication primitive (address information, User_Information) do not come from the AARQ APDU, but from the supporting layer frame carrying the AARQ APDU. In some communication profiles, the Service_Class parameter of the COSEM-OPEN service is linked to the frame type of the supporting layer. In some other communication profiles, it is linked to the response-allowed field of the xDLMS-Initiate.request APDU. See also Annex A.

NOTE 3 The ASEs only extract the parameters; their interpretation and the decision whether the proposed AA can be accepted or not is the job of the server AP.



5086

IEC 1122/13

5087 **Figure 39 – MSC for successful AA establishment preceded by a**
5088 **successful lower layer connection establishment**

5089 The server AP parses the fields of the AARQ APDU as described below.

5090 Fields of the Kernel functional unit:

- application-context-name: it carries the COSEM_Application_Context_Name the client proposes for the association;
- calling-AP-title: when the proposed application context uses ciphering, it shall carry the client system title. If a client system title has already been sent during a registration process, like in the S-FSK PLC profile, the calling-AP-title field shall carry the same system title. Otherwise, the AA shall be rejected and appropriate diagnostic information shall be sent;
- calling-AE-invocation-identifier: this field supports the client user identification process; see IEC 62056-6-2:2021, 4.4.2;
- calling-AE-qualifier: This field can be used to transport the public key certificate of the digital signature key of the client.

Fields of the authentication functional unit (when present):

- sender-acse-requirements:
  a) if not present or present but bit 0 = 0, then the authentication functional unit is not selected. Any following fields of the authentication functional unit may be ignored;
  d) if present and bit 0 = 1 then the authentication functional unit is selected;
- mechanism-name: it carries the COSEM_Authentication_Mechanism_Name the client proposes for the association;
- calling-authentication-value: it carries the authentication value generated by the client.

If the value of the mechanism-name or the calling-authentication-value fields are not acceptable then the proposed AA shall be refused.

When the parsing of the fields of the Kernel and the authentication functional unit is completed, the server continues with parsing the parameters of the xDLMS InitiateRequest APDU, carried by the user-information field of the AARQ:

- dedicated-key: it carries the dedicated key to be used in the AA being established;
- response-allowed: If the proposed AA is confirmed, the value of this parameter is TRUE (default), and the server shall send back an AARE APDU. Otherwise, the server shall not respond. See also Annex A.
- proposed-dlms-version-number, see 4.2.4;
- proposed-conformance, see 7.3.1;
- client-max-receive-pdu-sizes, see 4.2.4.

If all elements of the proposed AA are acceptable, the server AP invokes the COSEM-OPEN.response service primitive with the following parameters:

- Application_Context_Name: the same as the one proposed;
- Result: accepted;
- Failure_Type: Result-source: acse-service-user; Diagnostic: null;
- Responding_AP_title: if the negotiated application context uses ciphering, it shall carry the server system title. If a server system title has already been sent during a registration process, like in the case of the S-FSK PLC profile, the Responding_AP_Title parameter shall carry the same system title. Otherwise, the AA shall be aborted by the client;
- Responding_AE_Qualifier: This field can be used to transport the public key certificate of the digital signature key of the server;
- Fields of the AARE authentication functional unit:
  - (Responder_)ACSE_Requirements:

e) when no security (Lowest Level Security) authentication or Low Level Security (LLS) authentication is used, this field shall not be present, or if present, bit 0 (authentication) shall be set to 0. Any following fields of the authentication functional unit may be ignored;

f) when High Level Security (HLS) authentication is used, this field shall be present and bit 0 (authentication) shall be set to 1;

- Security_Mechanism_Name: it shall carry the COSEM_Authentication_Mechanism_ Name negotiated;
- Responding_Authentication_Value: it carries the authentication value generated by the server (StoC).

- Negotiated_xDLMS_context.

The CF assembles the AARE APDU – with the help of the xDLMS ASE and the ACSE – and sends it to the client AL via the supporting lower layer protocols, and enters the ASSOCIATED state. The proposed AA is established now, the server is able to receive xDLMS data transfer service request(s) – both confirmed and unconfirmed – and to send responses to confirmed service requests within this AA.

At the client side, the parameters of the AARE APDU received are extracted with the help of the ACSE and the xDLMS ASE, and passed to the client AP via the COSEM-OPEN.confirm service primitive. At the same time, the client AL enters the ASSOCIATED state. The AA is established now with the application context and xDLMS context negotiated.

If the application context proposed by the client is not acceptable or the authentication of the client is not successful, the COSEM-OPEN.response primitive is invoked with the following parameters:

- Application_Context_Name: the same as the one proposed, or the one supported by the server;
- Result: rejected-permanent or rejected-transient;
- Failure_Type: Result-source: acse-service-user; Diagnostic: an appropriate value;
- User_Information: an xDLMS InitiateResponse APDU with the parameters of the xDLMS context supported by the server.

If the application context proposed by the client is acceptable and the authentication of the client is successful but the xDLMS context cannot be accepted, the COSEM-OPEN.response primitive shall be invoked with the following parameters:

- Application_Context_Name: the same as the one proposed;
- Result: rejected-permanent or rejected-transient;
- Failure_Type: Result-source: acse-service-user; Diagnostic: no-reason-given;
- xDLMS_Initiate_Error, indicating the reason for not accepting the proposed xDLMS context.

In these two cases, upon invocation of the .response primitive, the CF assembles and sends the AARE APDU to the client via the supporting lower layer protocols. The proposed AA is not established, the server CF returns to the IDLE state.

At the client side, the parameters of the AARE APDU received are extracted with the help of the ACSE and the xDLMS_ASE, and passed to the client AP via the COSEM-OPEN.confirm primitive. The proposed AA is not established, the client CF returns to the IDLE state.

The server ACSE may not be capable of supporting the requested association, for example if the AARQ syntax or the ACSE protocol version are not acceptable. In this situation, it returns a COSEM-OPEN.response primitive to the client with an appropriate Result parameter. The Result Source parameter is appropriately assigned the symbolic value of "ACSE service-provider”. The COSEM-OPEN.indication primitive is not issued. The association is not established.

7.2.4.2 Repeated COSEM-OPEN service invocations

If a COSEM-OPEN.request primitive is invoked by the client AP referring to an already established AA, then the AL locally and negatively confirms this request with the reason that the requested AA already exists. Note, that this is always the case for pre-established AAs; see 7.2.4.4.

If, nevertheless, a server AL receives an AARQ APDU referencing to an already existing AA, it simply discards this AARQ, or, if it is implemented, it may also respond with the optional ExceptionResponse APDU.

7.2.4.3 Establishment of unconfirmed application associations

A client AP that desires to establish an unconfirmed AA, invokes the COSEM-OPEN.request primitive of the ASO with Service_Class == Unconfirmed. The response-allowed parameter of the xDLMS InitiateRequest APDU, carried by the user-information parameter of the AARQ is set to FALSE. The client AL does not wait any response from the server: the .confirm primitive is locally generated. Otherwise the procedure is the same as in the case of the establishment of confirmed AAs.

As the establishment of unconfirmed AAs does not require the server AP to respond to the association request coming from the client, in some cases – for example in the case of one-way communications or broadcasting – the establishment of unconfirmed AA is the only possibility.

After the establishment of an unconfirmed AA, xDLMS data transfer services using LN referencing can be invoked only in an unconfirmed manner, until the association is released. With SN referencing, only the UnconfirmedWrite service can be used.

7.2.4.4 Pre-established application associations

The purpose of pre-established AAs is to simplify data exchange. The AA establishment and release phases (phases 1 and 3 on Figure 4), using the COSEM-OPEN and COSEM-RELEASE services are eliminated and only data transfer services are used.

This International Standard does not specify how to establish such AAs: it can be considered, that this has already been done. Pre-established AAs can be considered to exist from the moment the lower layers are able to transmit APDUs between the client and the server.

As for all AAs, the logical devices contain an Association LN / SN interface object for the pre-established associations, too.

A pre-established AA can be either confirmed or unconfirmed (depending on the way it has been pre-established).

A pre-established AA cannot be released.

7.2.5 Protocol for application association release

7.2.5.1 Overview

An existing AA can be released gracefully or non-gracefully. Graceful release is initiated by the client AP. Non-graceful release takes place when an event unexpected by the AP occurs, for example a physical disconnection is detected.

7.2.5.2 Graceful release of an application association

DLMS®/COSEM provides two mechanisms to release AAs:

- by disconnecting the supporting layer of the AL;
- by using the ACSE A-Release service.

The first mechanism shall be supported in all profiles where the supporting layer of the AL is connection oriented.

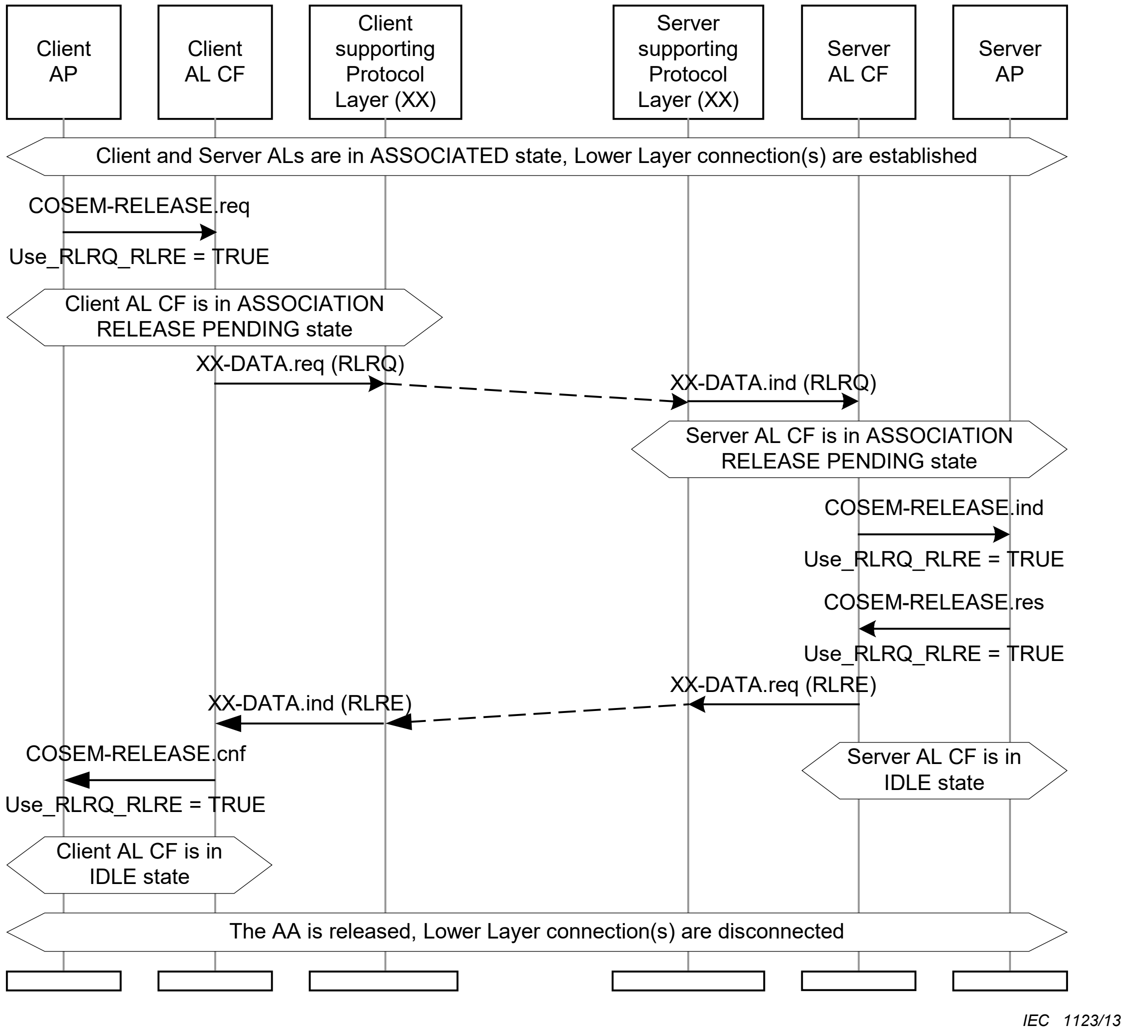
EXAMPLE The 3-layer, HDLC based, connection oriented profile.

To release an AA this way, the COSEM-RELEASE service shall be invoked with the Use_RLRQ_RLRE parameter not present or FALSE. Disconnecting the supporting layer shall release all AAs built on that supporting layer connection.

The second mechanism can be used to release an AA without disconnecting the supporting layer. It shall be supported in all profiles when the supporting layer is connectionless. It may be also used when the supporting layer is connection-oriented but the connection is not managed by the AL, or disconnection of the supporting layer is not practical because other applications may use it, or when there is a need to secure the COSEM-RELEASE service. It is the only way to release unconfirmed AAs.

To release an AA in this way, the COSEM-RELEASE service shall be invoked with the Use_RLRQ_RLRE parameter = TRUE. As specified in 6.3, the COSEM-RELEASE service can be secured by including a ciphered xDLMS InitiateRequest / InitiateResponse in the user-information field of the RLRQ / RLRE APDUs respectively, thus preventing a potential denial of service attack.

An example for releasing an AA using the ACSE A-RELEASE service is shown in Figure 40.



NOTE The release of an AA may require internal communication between the ASEs of the CF. These are not shown in Figure 40 to Figure 42.

Figure 40 – Graceful AA release using the A-RELEASE service

A client AP that desires to release an AA using the A-RELEASE service, shall invoke the COSEM-RELEASE.request service primitive with Use_RLRQ_RLRE == TRUE. The client CF enters the ASSOCIATION RELEASE PENDING state. It constructs then an RLRQ APDU and sends it to the server. If the AA to be released has been established with a ciphered context, then the user information field of the RLRQ APDU may contain a ciphered xDLMS InitiateRequest APDU, see 6.3.

When the server AL CF receives the RLRQ APDU, it checks first if the user-information field contains a ciphered xDLMS InitiateRequest APDU. If so, it tries to decipher it. If this is successful, it enters the ASSOCIATION RELEASE PENDING state and generates a COSEM-RELEASE.indication primitive with Use_RLRQ_RLRE == TRUE. Otherwise, it silently discards the RLRQ APDU and stays in the ASSOCIATED state.

The .response primitive is invoked by the server AP to indicate if the release of the AA is accepted or not, but only if the AA to be released is confirmed. Note, that the server AP cannot refuse a release request. Upon reception of the .response primitive the server AL CF constructs a RLRE APDU and sends it to the client. If the RLRQ APDU contained a ciphered xDLMS InititateRequest APDU then the RLRE APDU shall contain a ciphered xDLMS InitiateResponse APDU. The server AL CF returns to the IDLE state.

The .confirm primitive is generated by the client AL CF when the RLRE APDU is received. The supporting layer is not disconnected. The client AL CF returns to the IDLE state.

If the RLRE APDU received contains a ciphered xDLMS InitiateResponse APDU but it cannot be deciphered, then the RLRE APDU shall be discarded. It is left to the client to cope with the situation.

Figure 41 gives an example of graceful releasing a confirmed AA by disconnecting the corresponding lower layer connection.

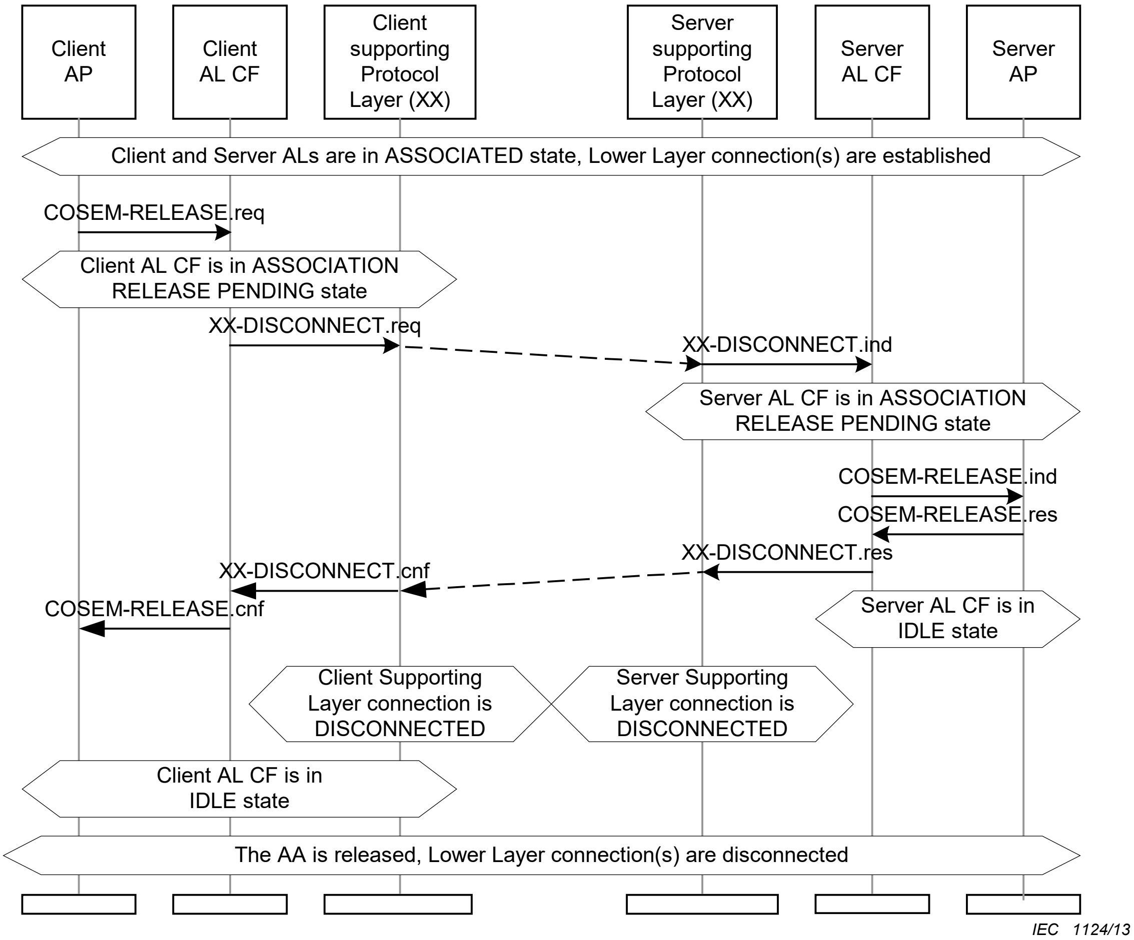


Figure 41 – Graceful AA release by disconnecting the supporting layer

A client AP that desires to release an AA not using the A-RELEASE service, invokes the COSEM-RELEASE.request primitive with Use_RLRQ_RLRE == FALSE or not present. The client AL CF enters the ASSOCIATION RELEASE PENDING state.

In communication profiles where the RLRQ service is mandatory, invoking the .request primitive with no Use_RLRQ_RLRE or with Use_RLRQ_RLRE == FALSE may lead to an error: the .request shall be locally and negatively confirmed. The client AL CF returns to the IDLE state.

When the client AL CF receives the .request primitive, it sends an XX-DISCONNECT.request primitive to the server.

When the server AL CF receives the XX-DISCONNECT.request primitive, the CF enters the ASSOCIATION RELEASE PENDING state. The COSEM-RELEASE.indication primitive is generated by the server AL CF with Use_RLRQ_RLRE == FALSE or not present.

The COSEM-RELEASE.response primitive is invoked by the server AP to indicate if the release of the AA is accepted or not. Note, that the server AP cannot refuse a release request. Upon reception of this primitive, the server AL CF sends an XX-DISCONNECT.response primitive to the client and returns to the IDLE state.

The COSEM-RELEASE.confirm primitive is generated by the client AL when the XX-DISCONNECT.confirm primitive is received. The supporting layer is disconnected. The client AL CF returns to the IDLE state.

7.2.5.3 Non-graceful release of an application association

Various events may result in a non-graceful release of an AA: detection of the disconnection of any lower layer connection (including the physical connection), detecting a local error, etc.

Non-graceful release – abort – of an AA is indicated to the COSEM AP with the help of the COSEM-ABORT service. The Diagnostics parameter of this service indicates the reason for the non-graceful AA release. The non-graceful release of AA is not selective: if it happens, all the existing association(s) – except the pre-established ones – shall be aborted.

Figure 42 shows the message sequence chart for aborting the AA, due to the detection of a physical disconnection.

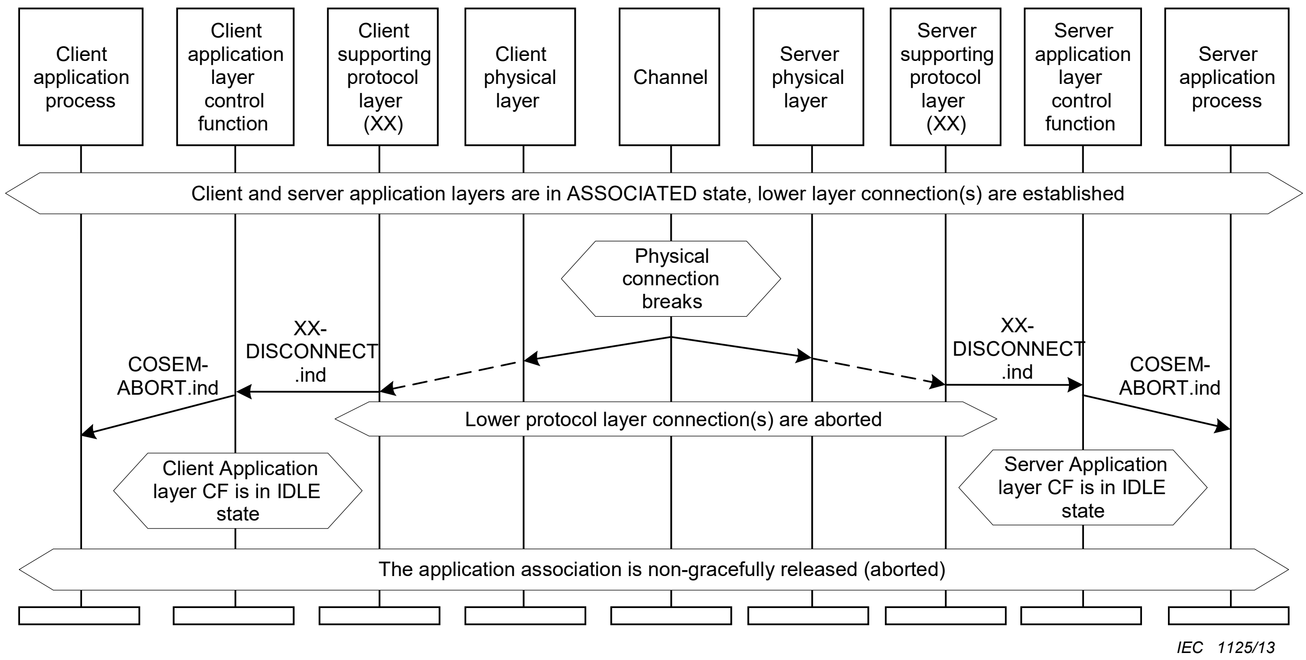


Figure 42 – Aborting an AA following a PH-ABORT.indication

7.3 Protocol for the data transfer services

7.3.1 Negotiation of services and options – the conformance block

The conformance block allows clients and servers using the same DLMS®/COSEM protocol, but supporting different capabilities to negotiate a compatible set of capabilities so that they can communicate. It is carried by the DLMS_Conformance parameter of the COSEM-OPEN service.

In DLMS®/COSEM none of the services or options are mandatory: the ones to be used are negotiated via the COSEM-OPEN service (the proposed-conformance parameter of the xDLMS InitiateRequest APDU and the negotiated-conformance parameter of the xDLMS InitiateResponse APDU). An implemented service shall be fully conforming to its specification.

If a service or option is not present in the negotiated conformance block, it may not be requested by the client.

The xDLMS conformance block can be distinguished from the DLMS® conformance block specified in IEC 61334-4-41:1996 by its tag: APPLICATION 31. It is shown in Table 67.

Table 67 – xDLMS Conformance block

Conformance block bit	Reserved	LN referencing	SN referencing
0	x		
1		general-protection [1]	general-protection [1]
2		general-block-transfer	general-block-transfer
3			read
4			write
5			unconfirmed-write
6		delta-value-encoding	delta-value-encoding
7	x		
8		attribute0-supported-with-set	
9		priority-mgmt-supported	
10		attribute0-supported-with-get	
11		block-transfer-with-get-or-read	block-transfer-with-get-or-read
12		block-transfer-with-set-or-write	block-transfer-with-set-or-write
13		block-transfer-with-action	
14		multiple-references	multiple-references
15			information-report
16		data-notification	data-notification
17		access	
18			parameterized-access
19		get	
20		set	
21		selective-access	
22		event-notification	
23		action	

[1] general-protection includes general-glo-ciphering, general-ded-ciphering, general-ciphering and general-signing

7.3.2 Confirmed and unconfirmed service invocations

7.3.2.1 Service invocations by the client

In general, data transfer services may be invoked in a confirmed or an unconfirmed manner. The time sequence of the service primitives corresponds to:

- Figure 35 item a) in case of confirmed service invocations; and
- Figure 35 item d) in case of unconfirmed service invocations.

A client AP that desires to access an attribute or a method of a COSEM interface object invokes the appropriate .request service primitive. The client AL constructs the APDU corresponding to the .request primitive and sends it to the server.

Upon the receipt of the .indication primitive, the server AP checks whether the service can be provided or not (validity, client access rights, availability, etc.). If it can, it locally applies the service required on the corresponding "real" object. In the case of confirmed services, the server AP invokes the appropriate .response primitive. The server AL constructs the APDU corresponding to the .response primitive and sends it to the client. The client AL generates the .confirm primitive.

If a confirmed service request cannot be processed by the server AL – for example the request has been received without establishing an AA first, or the request is otherwise erroneous – it is either discarded, or when possible, the server AL responds with a ConfirmedServiceError APDU, or, when implemented, with an ExceptionResponse APDU. These APDUs may contain diagnostic information about the reason of not being able to process the request. They are defined in Clause 8.

Within confirmed AAs, client/server type data transfer services can be invoked in a confirmed or unconfirmed manner.

Within unconfirmed AAs, client/server type data transfer services may be invoked in an unconfirmed manner only. With this, collisions due to potential multiple responses in the case of multicasting and/or broadcasting can be avoided.

In the case of unconfirmed services, three different kinds of destination addresses are possible: individual, group or broadcast. Depending on the destination address type, the receiving station shall handle incoming APDUs differently, as follows:

- XX-APDUs with an individual address of a COSEM logical device. If they are received within an established AA they shall be sent to the COSEM logical device addressed, otherwise they shall be discarded;
- XX-APDUs with a group address of a group of COSEM logical devices. These shall be sent to the group of COSEM logical devices addressed. However, the message received shall be discarded if there is no AA established between a client and the group of COSEM logical devices addressed;
- XX-APDUs with the broadcast address shall be sent to all COSEM logical devices addressed. However, the message received shall be discarded if there is no AA established between a client and the All-station address.

NOTE Unconfirmed AA-s between a client and a group of logical devices are established with a COSEM-OPEN service with Service_Class == Unconfirmed and a group of logical device addresses (for example broadcast address).

7.3.2.2 Service invocations by the server (unsolicited services)

The unsolicited services that may be invoked by the server are:

- InformationReport;
- EventNotification;
- DataNotification.

The InformationReport and the EventNotification services may be invoked only in an unconfirmed manner. The corresponding time sequence diagram is Figure 35 item d).

The DataNotification service may be invoked in three different ways:

1) unconfirmed, retry on supporting protocol layer failure;

2) unconfirmed, retry on missing supporting protocol layer confirmation;

3) confirmed, retry on missing confirmation.

The time sequence of the service primitives corresponds to:

in the case 1) Figure 35 item d);

in the case 2) Figure 35 item b);

in the case 3) Figure 35 item a).

The protocol of the DataNotification service is described in 6.10.

7.3.3 Protocol for the GET service

When the client AP desires to read the value of one or more COSEM interface object attributes, it uses the GET service.

As explained in 6.6, the encoded form of the request shall always fit in a single APDU.

On the other hand, the result may be too long to fit in a single APDU. In this case, either the service-specific or the general block transfer mechanism may be used. It is negotiated via bit 2 or bit 11 of the conformance block; see 7.3.1.

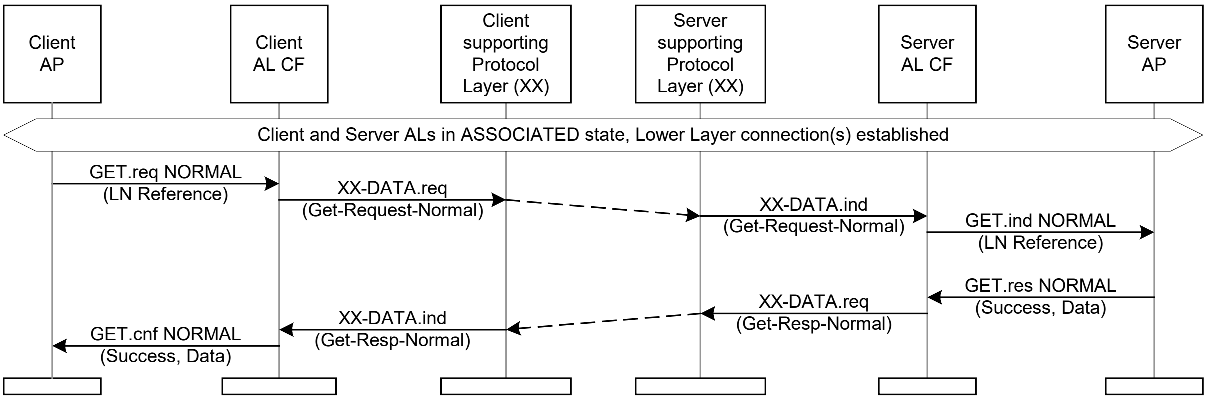
NOTE In some DLMS®/COSEM communication profiles segmentation is available to transfer long APDUs.

The GET service primitive types and the corresponding APDUs are shown in Table 68.

Table 68 – GET service types and APDUs

GET .req / .ind	Request APDU	Response APDU	GET .res / .cnf
NORMAL	Get-Request-Normal	Get-Response-Normal	NORMAL
		Get-Response-With-Datablock with Last-Block = FALSE	ONE-BLOCK
NEXT	Get-Request-Next	Get-Response-With-Datablock with Last-Block = FALSE	ONE-BLOCK
		Get-Response-With-Datablock with Last-Block = TRUE	LAST-BLOCK
WITH-LIST	Get-Request-With-List	Get-Response-With-List	WITH-LIST
		Get-Response-With-Datablock with Last-Block = FALSE	ONE-BLOCK

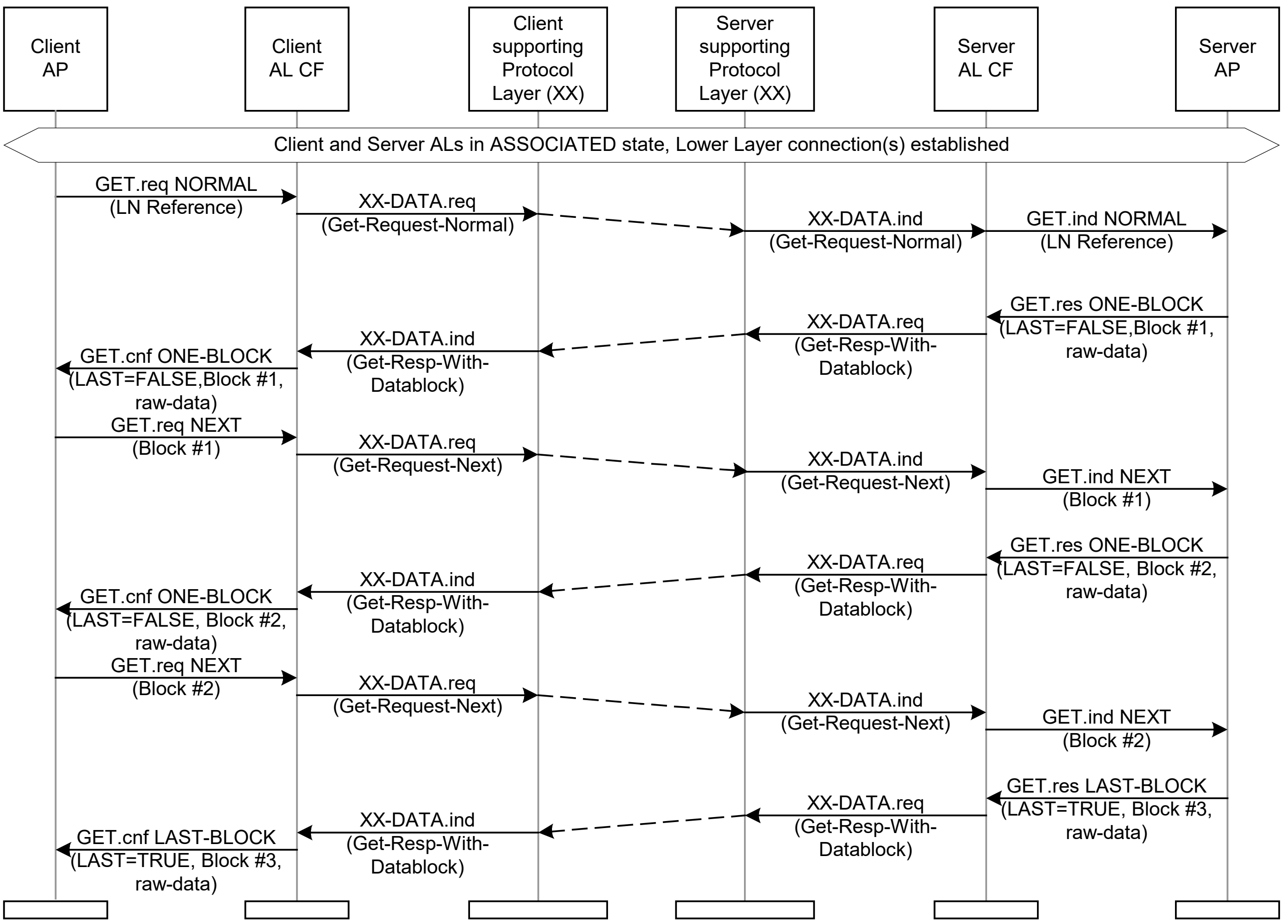
Figure 43 shows the MSC for a confirmed GET service in the case of success, without block transfer.



IEC 1126/13

Figure 43 – MSC of the GET service

Figure 44 shows the MSC of a confirmed GET service in the case of success, with the result returned in three blocks.



IEC 1127/13

Figure 44 – MSC of the GET service with block transfer

The GET.request primitive is invoked with Request_Type == NORMAL or WITH-LIST as appropriate. As in this case the data to be returned is too long to fit in a single APDU, the server AP sends it in blocks. First, the data is encoded, as if it would fit into a single APDU:

- if the value of a single attribute was requested, only the type and value shall be encoded (Data). If the Data cannot be delivered, the response shall be of type GET-NORMAL with Data_Access_Result;

- if the value of a list of attributes was requested, then the list of results shall be encoded: for each attribute, either Data or Data_Access_Result.

The result is a series of bytes, B_1, B_2, B_3,…., B_N. The server may generate the complete response upon receipt of the first GET.indication primitive or dynamically (on the fly).

The server AP assembles then a DataBlock_G structure:

- Last_Block == FALSE;
- Block_Number == 1;
- Result (Raw_Data) == the first K bytes of the encoded data: B_1, B_2, B_3,…., B_K.

It is recommended to start the numbering of the blocks from 1.

The server AP invokes then the GET-RESPONSE-ONE-BLOCK service primitive with Response_Type == ONE-BLOCK carrying this DataBlock-G structure.

Upon reception of the .response primitive, the server AL builds a Get-Response-With-Datablock APDU carrying the parameters of the .response primitive and sends it to the client.

Upon reception of this APDU, the client AL generates a .confirm primitive with Response_Type == ONE-BLOCK. The client AP is informed now that the response is provided in several blocks. It stores the data block received (B_1, B_2, B_3,…., B_K), then acknowledges its reception and asks for the next one by invoking a GET-REQUEST-NEXT service primitive. The block number shall be the same as the block number of the data block received. The client AL builds a Get-Request-Next APDU and sends it to the server.

When the server AL invokes the GET.indication primitive with Request_Type == NEXT, the server AP prepares and sends the next data block, including B_{K+1}, B_{K+2}, B_{K+3},…., B_L, with block-number == 2. This exchange of sending data blocks and acknowledgements continues until the last data block, carrying B_M, B_{M+1}, B_{M+2},…., B_N is sent. The last GET.response primitive is invoked with Response_Type == LAST-BLOCK: in the DataBlock-G structure Last_Block == TRUE. This last data block is not acknowledged by the client.

Throughout the whole procedure, the Invoke_Id and the Priority parameters shall be the same in each primitive.

If during a long data transfer the server receives another service request, it is served according to the priority rules and the priority management settings (Conformance block bit 9).

If any error occurs during the long data transfer, the transfer is aborted. The error cases are:

a) The server is not able to provide the next block of data for any reason. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock_G structure with:

  - Last_Block == TRUE;
  - Block_Number == number of the block confirmed by the client +1;
  - Result == Data_Access_Result, indicating the reason of the failure.

b) The Block_Number parameter in a GET-REQUEST-NEXT service primitive is not equal to the number of the previous block sent by the server. The server interprets this, as if the client would like to abort the ongoing transfer. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock_G structure with:

  - Last_Block == TRUE;

- Block_Number == equal to the block number received from the client;
- Result == Data-Access-Result, long-get-aborted.

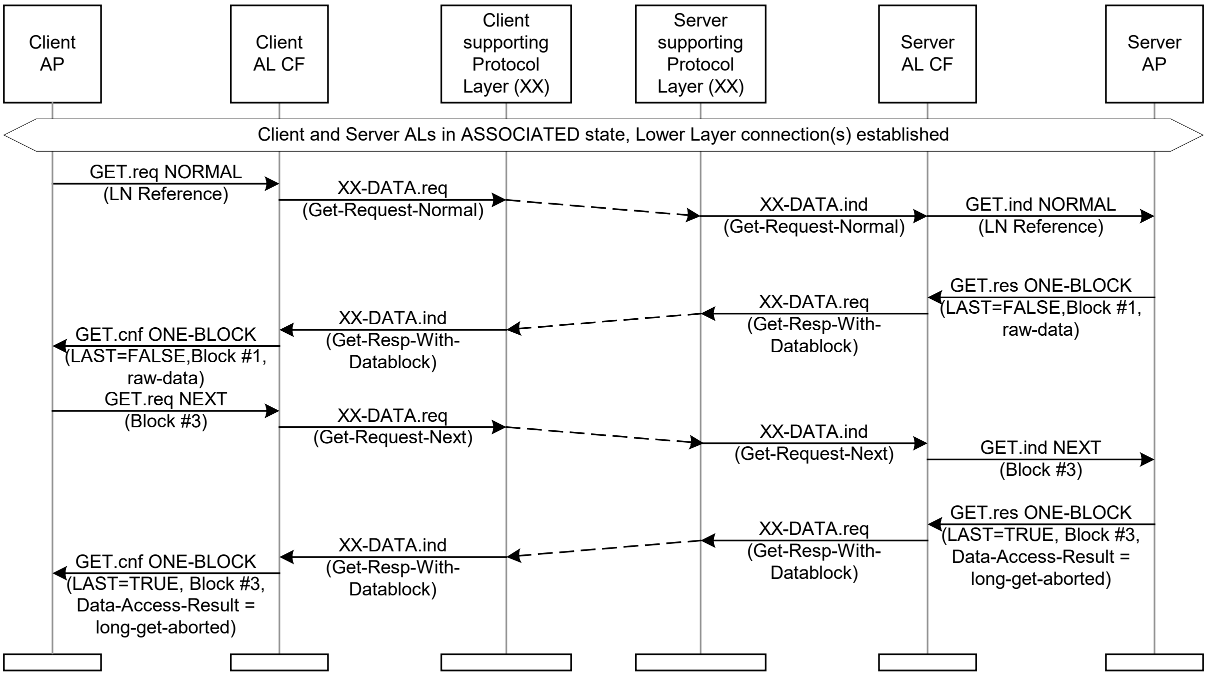
c) The server may receive a Get-Request-Next APDU when no long data transfer is in progress. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock_G structure with:

- Last-block == TRUE;
- Block-Number == equal to the block number received from the client;
- Result == Data-Access-Result, no-long-get-in-progress.

d) The block number sent by the server is not equal to the next in sequence. In this case, the client shall abort the block transfer (see case b).

If, in the error cases above, the server is not able to invoke a GET-RESPONSE-LAST-BLOCK service primitive for any reason, it shall invoke a GET-RESPONSE-NORMAL service primitive, with the Data-Access-Result parameter indicating the reason of the failure. The server shall send a Get-Response-Normal APDU.

The MSC for error case b), long get aborted, is shown in Figure 45:



IEC 1128/13

Figure 45 – MSC of the GET service with block transfer, long GET aborted

7.3.4 Protocol for the SET service

When the client AP desires to write the value of one or more COSEM interface object attributes, it uses the SET service.

As explained in 6.7, the encoded form of the request may fit in a single request or not. In this latter case, either the service-specific or the general block transfer mechanism may be used. It is negotiated via bit 2 or bit 12 of the conformance block; see 7.3.1.

NOTE In some DLMS®/COSEM communication profiles segmentation is available to transfer long APDUs.

The SET service primitive types and the corresponding APDUs are shown in Table 69.

Table 69 – SET service types and APDUs

<table>
<tr><th>SET .req / .ind</th><th>Request APDU</th><th>Response APDU</th><th>SET .res / .cnf</th></tr>
<tr><td>NORMAL</td><td>Set-Request-Normal</td><td>Set-Response-Normal</td><td>NORMAL</td></tr>
<tr><td>FIRST-BLOCK</td><td>Set-Request-With-First-Datablock
Last-Block = FALSE</td><td rowspan="2">Set-Response-Datablock</td><td rowspan="2">ACK-BLOCK</td></tr>
<tr><td>ONE-BLOCK</td><td>Set-Request-With-Datablock
Last-Block = FALSE</td></tr>
<tr><td rowspan="2">LAST-BLOCK</td><td rowspan="2">Set-Request-With-Datablock
Last-Block = TRUE</td><td rowspan="2">Set-Response-Last-Datablock</td><td>LAST-BLOCK</td></tr>
<tr><td>LAST-BLOCK-WITH-LIST</td></tr>
<tr><td>WITH-LIST</td><td>Set-Request-With-List</td><td>Set-Response-With-List</td><td>WITH-LIST</td></tr>
<tr><td>FIRST-BLOCK-WITH-LIST</td><td>Set-Request-With-List-And-With-First-Datablock</td><td>Set-Response-Datablock</td><td>ACK-BLOCK</td></tr>
</table>

Figure 46 shows the MSC of a confirmed SET service, in the case of success, without block transfer.

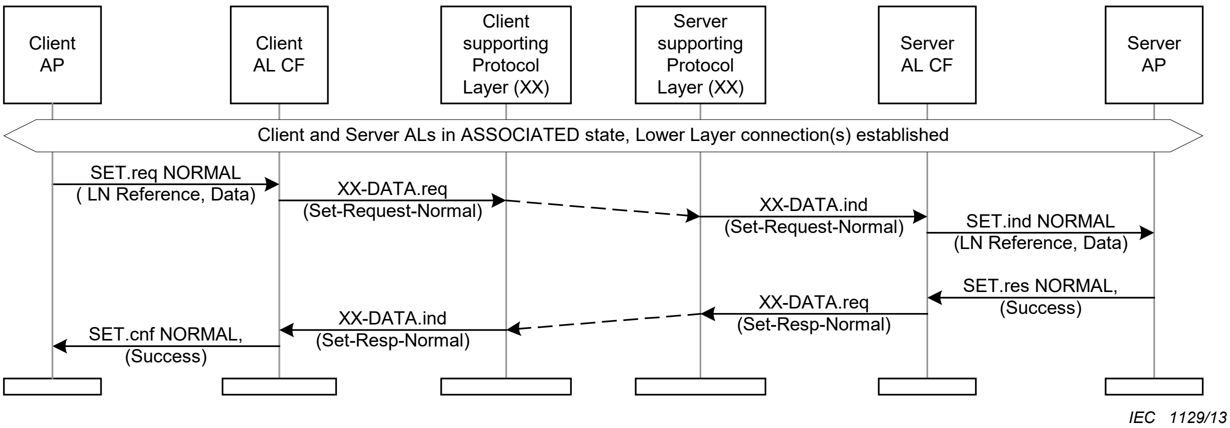


Figure 46 – MSC of the SET service

Figure 47 shows the MSC of a confirmed SET service in the case of success, with the request sent in three blocks.

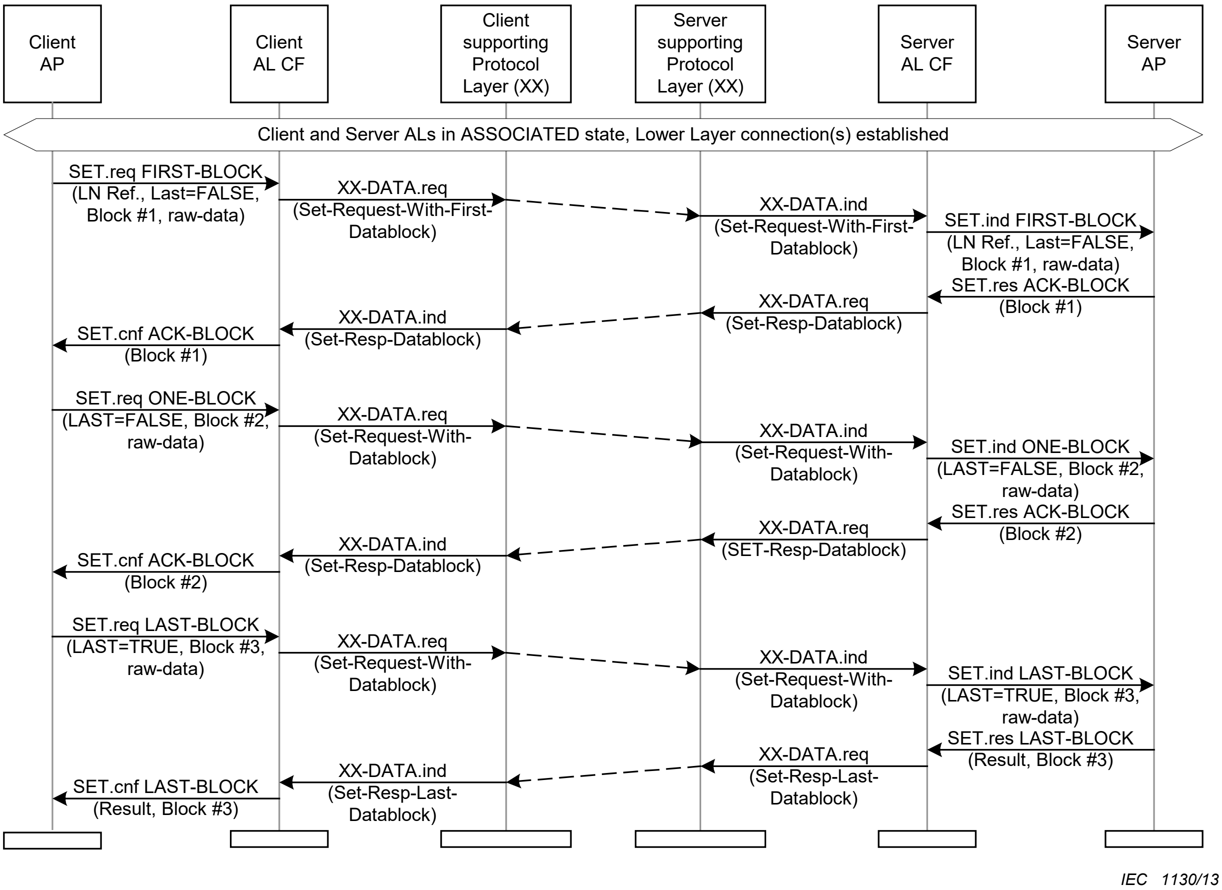


Figure 47 – MSC of the SET service with block transfer

As in this case the data to be sent is too long to fit in a single APDU in this case, the client AP sends it in blocks. First, the data is encoded as if it would fit in a single APDU. The result is a series of bytes, B_1, B_2, B_3,…., B_N. The client may generate the complete request (B_1, B_2, B_3,….B_N) in one step or dynamically (on the fly).

The client AP assembles then a DataBlock_SA structure:

- Last_Block == FALSE;
- Block_Number == 1;
- Raw_Data == the first K bytes of the encoded data: B_1, B_2, B_3,…., B_K.

The client AP invokes then the SET-REQUEST-FIRST-BLOCK or SET-REQUEST-FIRST-BLOCK-WITH-LIST service primitive as appropriate, carrying the attribute reference(s) and this DataBlock-SA structure.

Upon reception of the .request primitive, the client AL builds the appropriate Set-Request APDU carrying the parameters of the .request primitive and sends it to the server.

The server stores the data block received, then acknowledges its reception and asks for the next one by invoking a SET.response primitive with Response_Type == ACK-BLOCK and with the same block number as the number of the block received.

To send the next data block carrying B_{K+1}, B_{K+2}, B_{K+3},…., $B_{L,}$ the client AP invokes a SET-REQUEST-ONE-BLOCK service primitive. This exchange of sending data blocks and acknowledgements continues until the last data block carrying B_M, B_{M+1}, B_{M+2},…., B_N is sent, by invoking a SET-REQUEST-LAST-BLOCK service primitive with Last_Block == TRUE.

When these primitives are invoked, the client AL builds a Set-Request-With-Datablock APDU, carrying a DataBlock_SA structure and sends these APDUs to the server.

When the server AP receives the last datablock, it invokes a SET-RESPONSE-LAST-BLOCK or SET-RESPONSE-LAST-BLOCK-WITH-LIST service primitive as appropriate. The Result parameter carries the result of the complete SET service invocation. The Block_Number parameter confirms the reception of the last block.

Throughout the whole procedure, the Invoke_Id and the Priority parameters shall be the same in each primitive.

If during a long data transfer the server receives another service request, it is served according to the priority rules and the priority management settings (Conformance block bit 9).

If any error occurs during the long data transfer, the transfer is aborted. The error cases are:

a) The server is not able to handle the data block received, for any reason. In this case, the server AP shall invoke a SET-RESPONSE-LAST-BLOCK or SET-RESPONSE-LAST-BLOCK-WITH-LIST service primitive as appropriate. The Result parameter indicates the reason for aborting the transfer;

g) The Block_Number parameter in a SET-REQUEST-ONE-BLOCK service primitive is not equal to the block number expected by the server (last received + 1). The server interprets this as if the client would like to abort the ongoing transfer. In this case, the server AP shall invoke a SET-RESPONSE-LAST-BLOCK or SET-RESPONSE-LAST-BLOCK-WITH-LIST service primitive as appropriate with the Result parameter Data_Access_Result == long-set-aborted;

h) The server may receive a Set-Request-With-Datablock APDU when no long data transfer is in progress. In this case, the server AP shall invoke a SET-RESPONSE-LAST-BLOCK service primitive with the Result parameter Data_Access_Result == no-long-set-in-progress.

If, in the error cases above, for any reason the server is not able to invoke a SET-RESPONSE-LAST-BLOCK service primitive, it invokes a SET-RESPONSE-NORMAL service primitive with the Data-Access-Result parameter indicating the reason of the failure.

7.3.5 Protocol for the ACTION service

When the client AP desires to invoke one or more COSEM interface objects methods, it uses the ACTION service. As explained in 6.8, the ACTION service comprises two phases.

If the method references and method invocation parameters or the return parameters do not fit in a single APDU, either the service-specific or the general block transfer mechanism may be used. It is negotiated via bit 2 or bit 13 of the conformance block; see 7.3.1.

NOTE In some DLMS®/COSEM communication profiles segmentation is available to transfer long APDUs.

The ACTION service primitive types and the corresponding APDUs are shown in Table 70.

Table 70 – ACTION service types and APDUs

ACTION .req / .ind	Request APDU	Response APDU	SET .res / .cnf
NORMAL	Action-Request-Normal	Action-Response-Normal	NORMAL
		Action-Response-With-Pblock	ONE-BLOCK
NEXT	Action-Request-Next-Pblock	Action-Response-With-Pblock	ONE-BLOCK
		Action-Response-With-Pblock	LAST-BLOCK
FIRST-BLOCK	Action-Request-With-First-Pblock	Action-Response-Next-Pblock	NEXT
ONE-BLOCK	Action-Request-With-Pblock		
LAST-BLOCK	Action-Request-With-Pblock	Action-Response-Normal	NORMAL
		Action-Response-With-Pblock	ONE-BLOCK
WITH-LIST	Action-Request-With-List	Action-Response-With-List	WITH-LIST
		Action-Response-With-Pblock	ONE-BLOCK
WITH-LIST-AND-FIRST-BLOCK	Action-Request-With-List-And-With-First-Pblock	Action-Response-Next-Pblock	NEXT

Figure 48 illustrates the MSC of a confirmed ACTION service in the case of success, without block transfer.

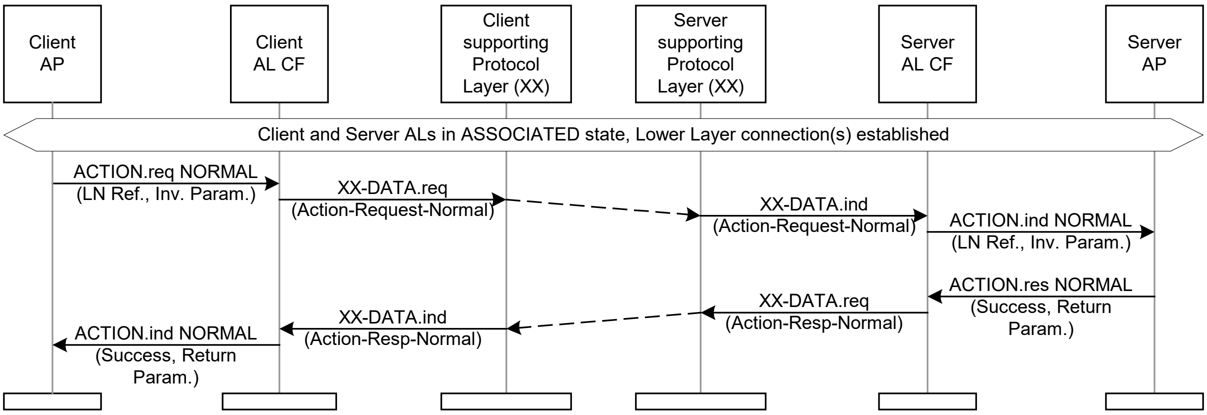


Figure 48 – MSC of the ACTION service *IEC 1131/13*

The ACTION service can transport data in both directions:

- in the first phase, the client sends the ACTION.request with the method invocation parameters for the method(s) referenced and the server acknowledges them. The process is essentially the same as in the case of the SET service;
- in the second phase, the server sends the ACTION.response with the result of invoking the method(s) and the return parameters. The process is essentially the same as in the case of the GET service.

Throughout the whole procedure, the Invoke_Id and the Priority parameters shall be the same in each primitive.

If during a long data transfer the server receives another service request, it is served according to the priority rules and the priority management settings (Conformance block bit 9).

Figure 49 illustrates the MSC in the case when block transfer takes place in both directions.

If any error occurs during the long data transfer, the transfer shall be aborted. Error cases are the same as in the case of the GET and SET services.

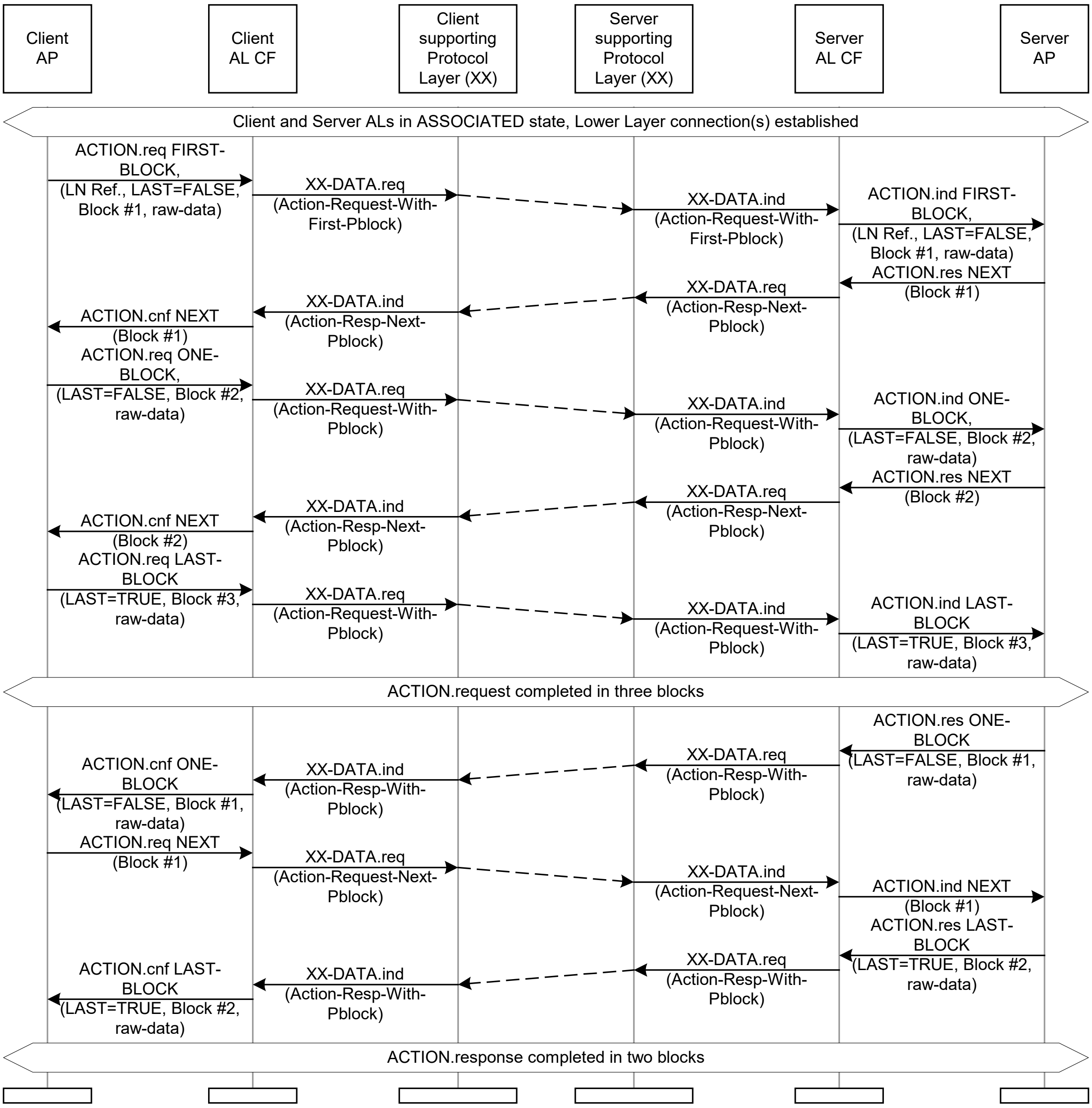


Figure 49 – MSC of the ACTION service with block transfer *IEC 1132/13*

7.3.6 Protocol for the ACCESS service

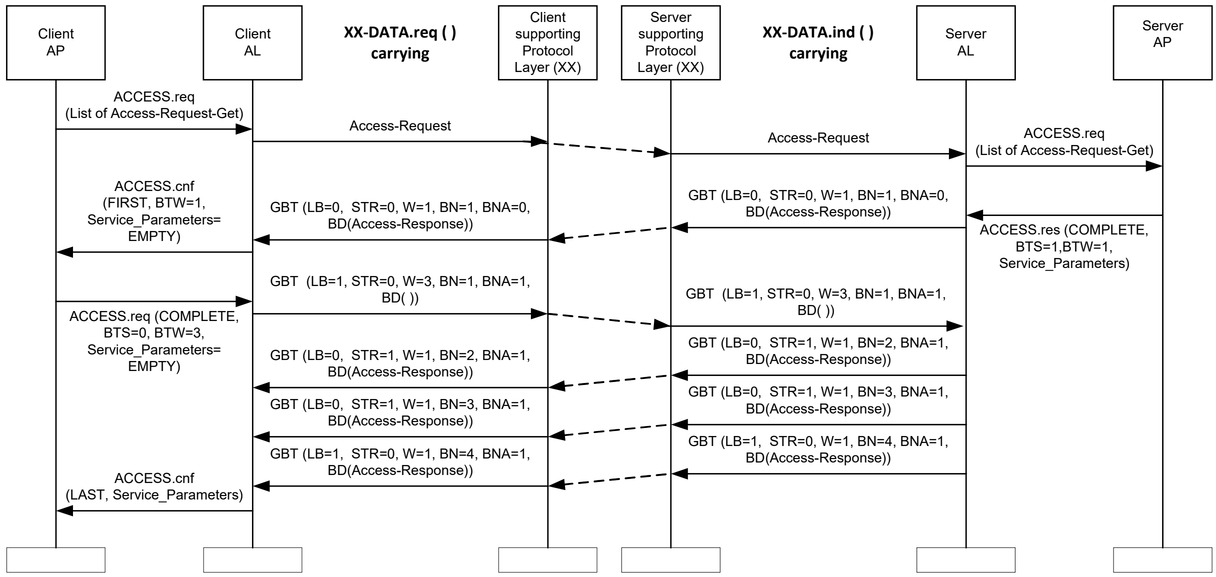
The client can use the ACCESS service to read or write the value of one or more COSEM object attributes or to invoke one or more methods.

The protocol of the ACCESS service is specified by way of message sequence charts, including cases where it is used together with general block transfer and general message protection.

NOTE See also 4.2.4.4.7, 6.5 and 7.3.13.

Figure 50 shows the MSC of an ACCESS service used to get one COSEM object attribute values. The request fits in a single APDU. The response is long therefore the server sends back the response using the GBT mechanism: portions of the access-response APDU are carried by the block-data field of general-block-transfer APDUs.

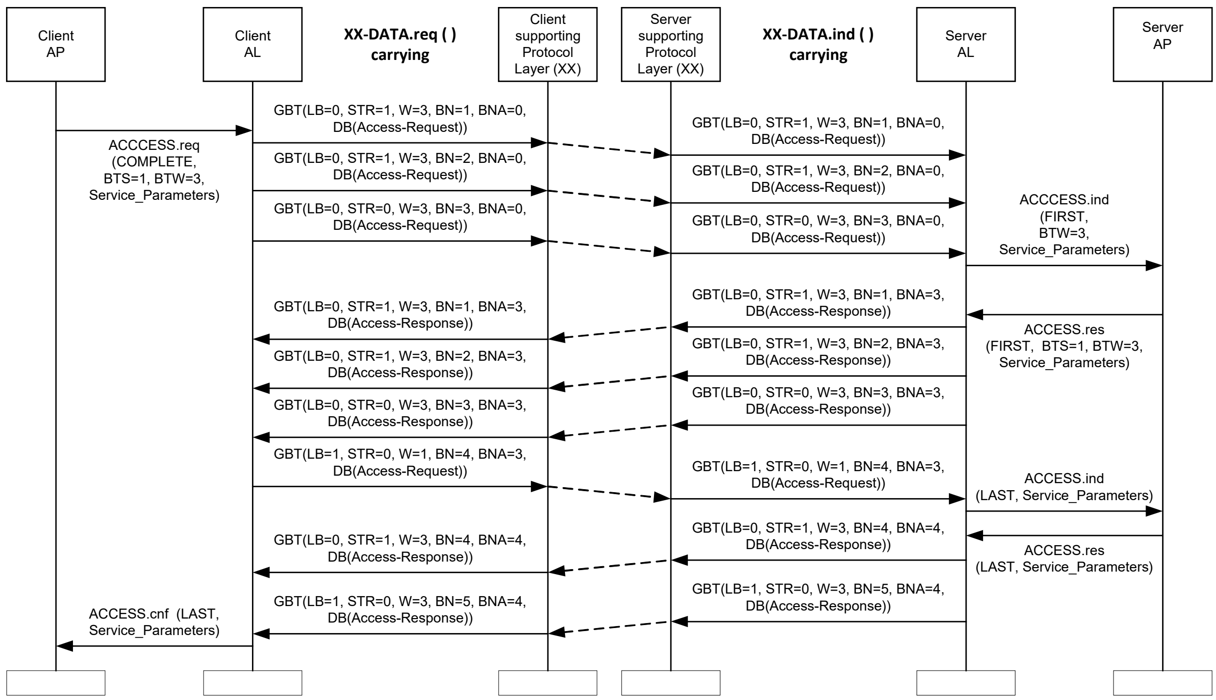
When the client receives the first general-block-transfer APDU, it switches to GBT announcing that streaming with window size 3 is supported.



BTS: Block_Transfer_Streaming, BTW: Block_Transfer_Window,
GBT: General-Block-Transfer APDU, LB: Last block, STR: Streaming, W: Window, BN: Block number, BNA: Block number acknowledged, BD: Block data

Figure 50 – Access Service with long response

Figure 51 shows the MSC of an ACCESS service used to carry a list of requests, which does not fit in a single APDU, therefore GBT is used. The response is also long therefore the server also uses GBT. Both parties know a priori that the other party supports streaming with window size = 3.



BTS: Block_Transfer_Streaming, BTW: Block_Transfer_Window,
GBT: General-Block-Transfer APDU, LB: Last block, STR: Streaming, W: Window, BN: Block number, BNA: Block number acknowledged, BD: Block data

Figure 51 – Access Service with long request and response

7.3.7 Protocol of the DataNotification service

When the server AP invokes a DataNotification.request service primitive, the server AL builds the DataNotification APDU and passes it to the supporting protocol layer.

When the client AL receives this APDU, it invokes the DataNotification.indication service primitive.

In case 1), unconfirmed, retry on supporting protocol layer failure (see 7.3.2.2):

- the push operation is deemed as failed when the server AP receives a DataNotification.cnf service primitive with Service_Class == Unconfirmed and Result == LOWER_LAYER_FAILED. In this case, the server AP may attempt a retry after a repetition delay;
- the push operation is deemed as successful when the server AP receives a DataNotification.cnf with Service_Class == Unconfirmed and Result == CONFIRMED;

Figure 52 shows the MSC of the DataNotification service for this case.

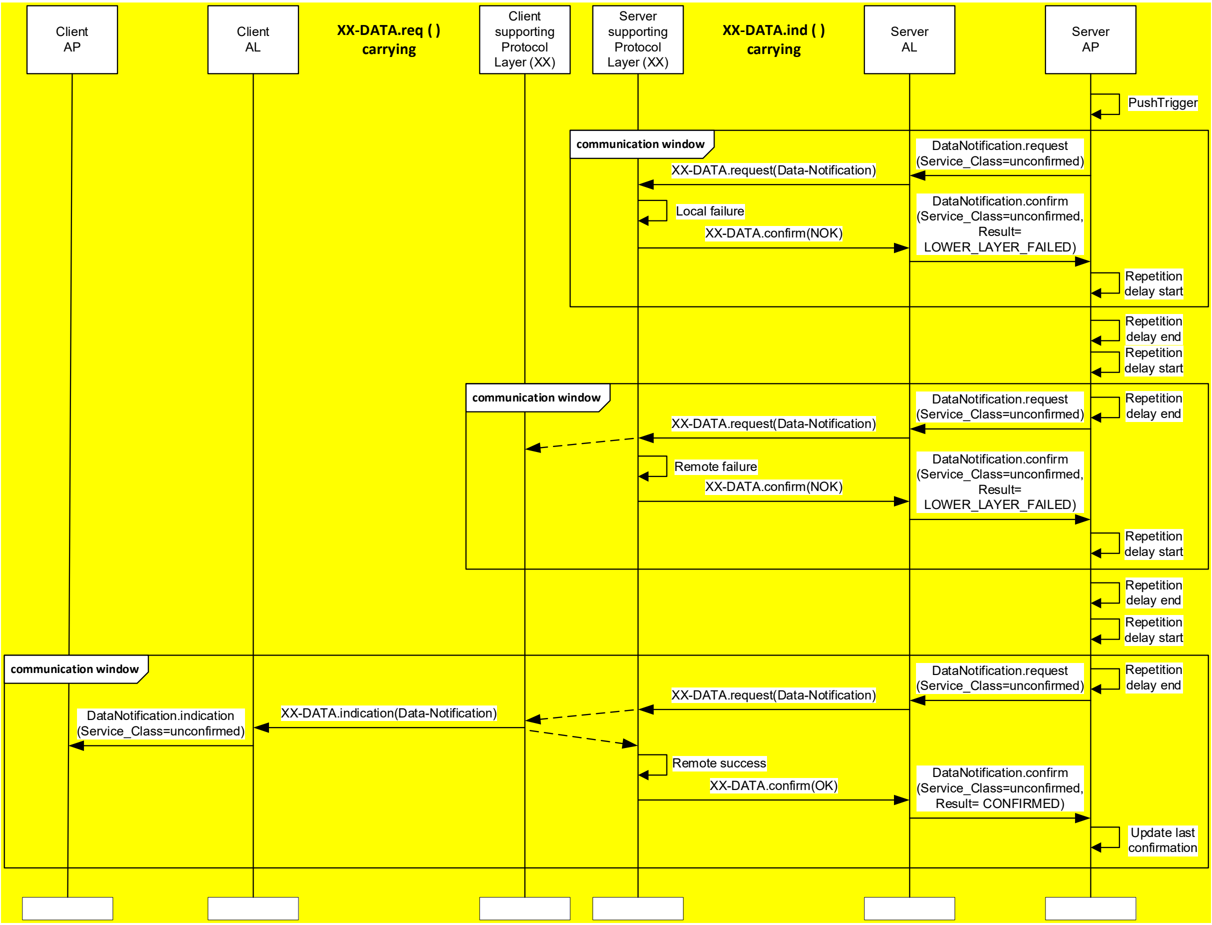


Figure 52 – MSC for the DataNotification service, case 1)

In case 2), unconfirmed, retry on missing supporting protocol layer confirmation (see 7.3.2.2):

- the push operation is deemed as failed if the repetition delay expires before a confirmation is received. In this case, the server may attempt a retry;

- the push operation is deemed as successful when the server supporting protocol layer informs the AL that the DataNotification APDU has been successfully received by the remote supporting protocol layer. In this case, the server AL invokes a DataNotification.cnf service primitive with Service_Class == Unconfirmed and Result == CONFIRMED.

Figure 53 shows the MSC of the DataNotification service for this case.

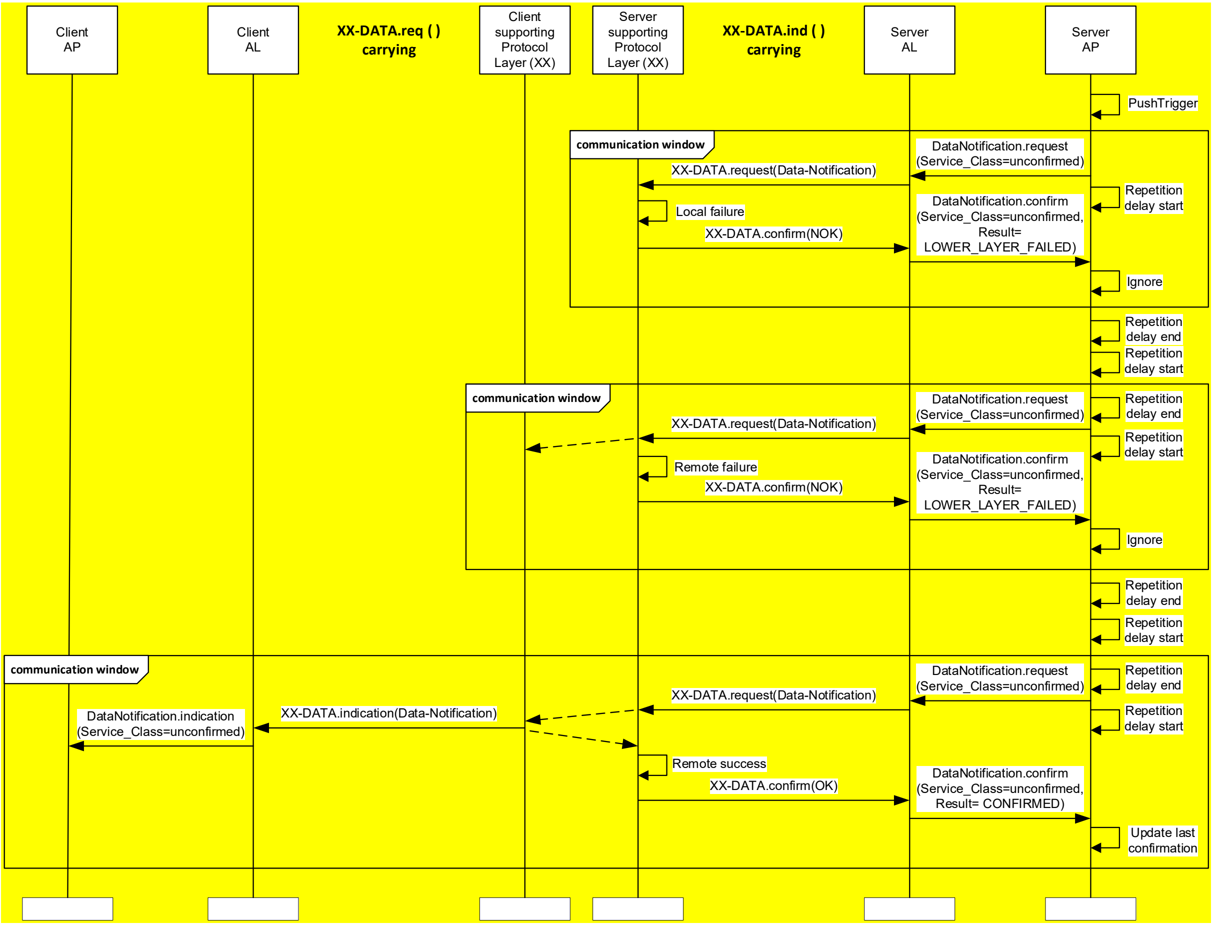


Figure 53 – MSC for the DataNotification service, case 2)

In case 3), confirmed, retry on missing confirmation (see 7.3.2.2):

- when the client AL receives a correct DataNotification APDU, it invokes the DataNotification.ind service primitive. If the Data-Notification APDU cannot be processed for any reason, it is discarded;
- the client AP invokes the DataNotification.response service primitive with Result == Confirmed. The client AL builds the Data-Notification-Confirm APDU and sends it to the server;
- the push operation is deemed as failed if the repetition delay expires before a confirmation is received. In this case, the server may attempt a retry;
- the push operation is deemed as successful when the AL layer receives the Data-Notification-Confirm APDU and invokes the DataNotification.cnf service primitive with Service_Class == Confirmed and Result == CONFIRMED.

Figure 54 shows the MSC of the DataNotification service for this case.

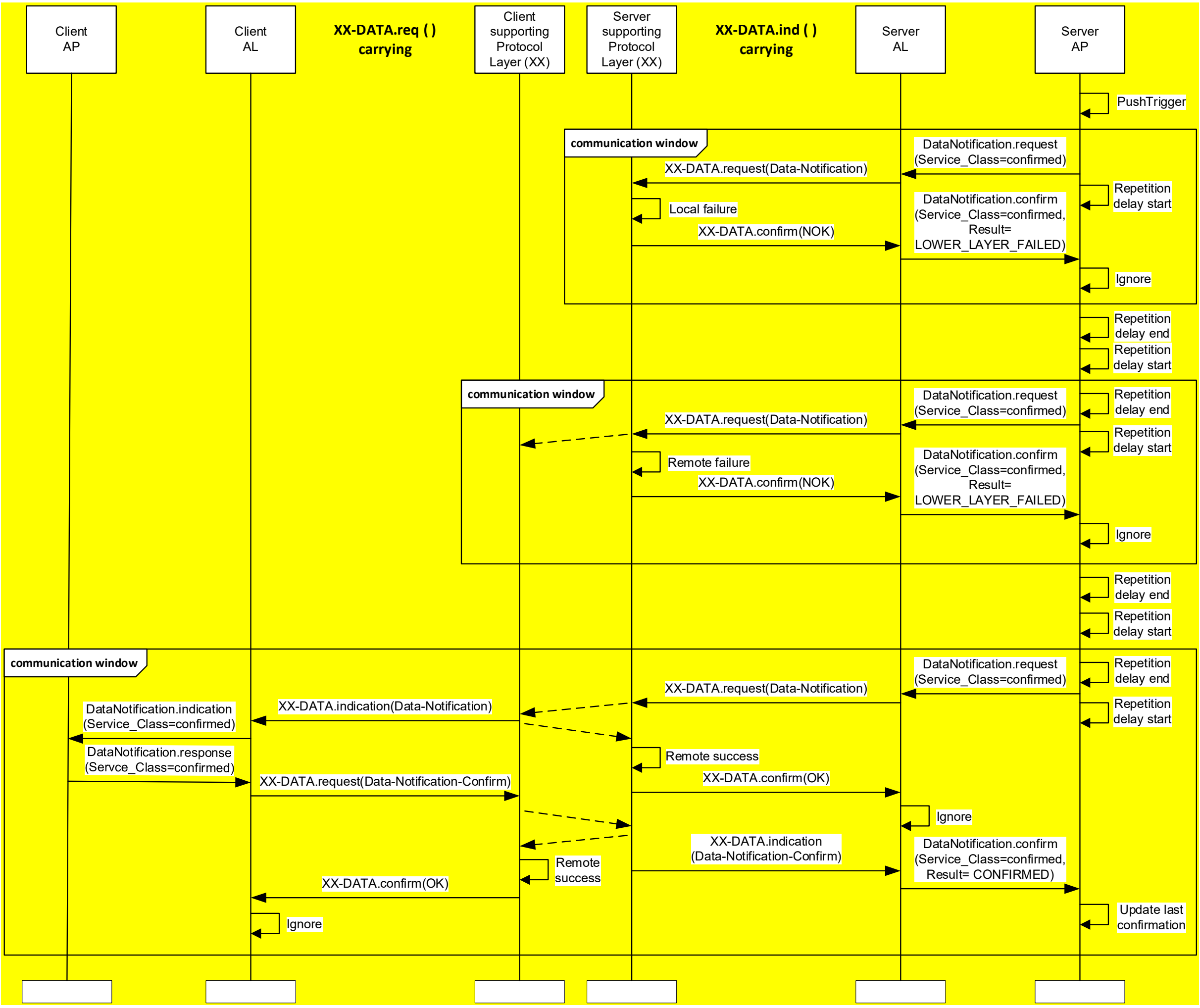


Figure 54 – MSC for the DataNotification service, case 3)

7.3.8 Protocol for the EventNotification service

Upon invocation of the EventNotification.request service, the Server AL builds an EventNotificationRequest APDU. The possibilities to send out this APDU depend on the communication profile and the connection status of the lower layers. Therefore, the protocol of the EventNotification service is further discussed in the parts of IEC 62056 describing the communication profiles;

In any case, in order to send the value(s) of attribute(s) to the client, without the client requesting it:

- the server uses the EventNotification.request service primitive;
- upon invocation of this primitive, the server AL builds an EventNotificationRequest APDU;
- this APDU is carried by the supporting layer service at the first opportunity to the client. The service type and the availability of this first opportunity depends on the communications profile used;
- upon reception of the EventNotificationRequest APDU, the client AL generates an EventNotification.indication primitive to the COSEM client AP;

NOTE At the client side, it is always EventNotification.indication, independently of the referencing scheme (LN or SN) used by the server.

- by default, event notifications are sent from the management logical device (server) to the management AP (client).

7.3.9 Protocol for the Read service

As explained in 6.14, the Read service is used when the server uses SN referencing, either to read (a) COSEM interface object attribute(s), or to invoke (a) method(s) when return parameters are expected:

- in the first case, the GET.request service primitives are mapped to Read.request primitives and the Read.confirm primitives are mapped to GET.confirm primitives. The mapping and the corresponding SN APDUs are shown in Table 71;
- in the second case, the ACTION.request service primitives are mapped to Read.request primitives and the Read.response primitives are mapped to ACTION.response primitives. The mapping and the corresponding SN APDUs is shown in Table 72.

NOTE In the mapping tables below, the following notation is used:

- for LN services, only the request and response types are shown without service parameters;
- for SN services, the name of the service primitive is followed by the service parameters in brackets. Service parameter name elements are capitalized and joined with an underscore to signify a single entity. Parameters that may be repeated are shown in curly brackets. The choices that can be taken for the Variable_Access_Specification parameter are listed following the symbol "=". Alternatives are separated by the vertical bar "I";
- for SN APDUs, the name of the APDU is followed by the symbol "::=" and the fields in brackets. The field name elements are not capitalized and are joined with a dash to signify a single entity. Fields that may be repeated are shown in curly brackets. Alternatives are separated by the vertical bar "I".

Table 71 – Mapping between the GET and the Read services

From GET.request of type	To Read.request	SN APDU
NORMAL	Read.request (Variable_Access_Specification) Variable_Access_Specification = Variable_Name I Parameterized_Access;	ReadRequest::= (variable-name I parameterized-access)
NEXT	Read.request (Variable_Access_Specification) Variable_Access_Specification = Block_Number_Access;	ReadRequest::= (block-number-access)
WITH-LIST	Read.request ({Variable_Access_Specification}) Variable_Access_Specification = Variable_Name I Parameterized_Access;	ReadRequest::= ({variable-name I parameterized-access})
To GET.confirm of type	**SN APDU**	**From Read.response**
NORMAL	ReadResponse::= (data I data-access-error)	Read.response (Data I Data_Access_Error)
ONE-BLOCK	ReadResponse::= (data-block-result)	Read.response (Data_Block_Result) with Last_Block = FALSE
LAST-BLOCK	ReadResponse::= (data-block-result)	Read.response (Data_Block_Result) with Last_Block = TRUE
WITH-LIST	ReadResponse::= ({data I data-access-error})	Read.response ({Data I Data_Access_Error})

Table 72 – Mapping between the ACTION and the Read services

From ACTION.request of type	To Read.request	SN APDU
NORMAL	Read.request (Variable_Access_Specification) Variable_Access_Specification = Parameterized_Access; with Variable_Name = method reference, Selector = 0, Parameter = method invocation parameter or null-data	ReadRequest::= (parameterized-access)
NEXT	Read.request (Variable_Access_Specification) Variable_Access_Specification = Block_Number_Access;	ReadRequest:: = (block-number-access)
FIRST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Raw_Data = one part of the method reference(s) and method invocation parameter	ReadRequest::= (read-data-block-access)
ONE-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = FALSE, Block_Number = next number, Raw_Data = as above	ReadRequest::= (read-data-block-access)
LAST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = TRUE, Block_Number = next number, Raw_Data = as above	ReadRequest::= (read-data-block-access)
WITH-LIST	Read.request ({Variable_Access_Specification}) Variable_Access_Specification = Parameterized_Access; with Variable_Name = method reference, Selector = 0, Parameter = method invocation parameter or null-data	ReadRequest::= ({parameterized-access})
WITH-LIST-AND-FIRST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Raw_Data = as above	ReadRequest::= (read-data-block-access)
To ACTION.confirm	**SN APDU**	**From Read.response**
NORMAL	ReadResponse::= (data I data-access-error)	Read.response (Read_Result) Read_Result = Data I Data_Access_Error;

From ACTION.request of type	To Read.request	SN APDU
ONE-BLOCK	ReadResponse::= (data-block-result)	Read.response (Read_Result) Read_Result = Data_Block_Result; with Last_Block = FALSE
LAST-BLOCK	ReadResponse::= (data-block-result)	Read.response (Read_Result) Read_result = Data_Block_Result; with Last_Block = TRUE
NEXT	ReadResponse::= (block-number)	Read.confirm (Read_Result) Read_Result = Block_Number;
WITH-LIST	ReadResponse::= ({data I data-access-error})	Read.response ({Read_Result}) Read_Result = Data I Data_Access_Error;

Figure 55shows the MSC of a Read service used to read the value of a single attribute.

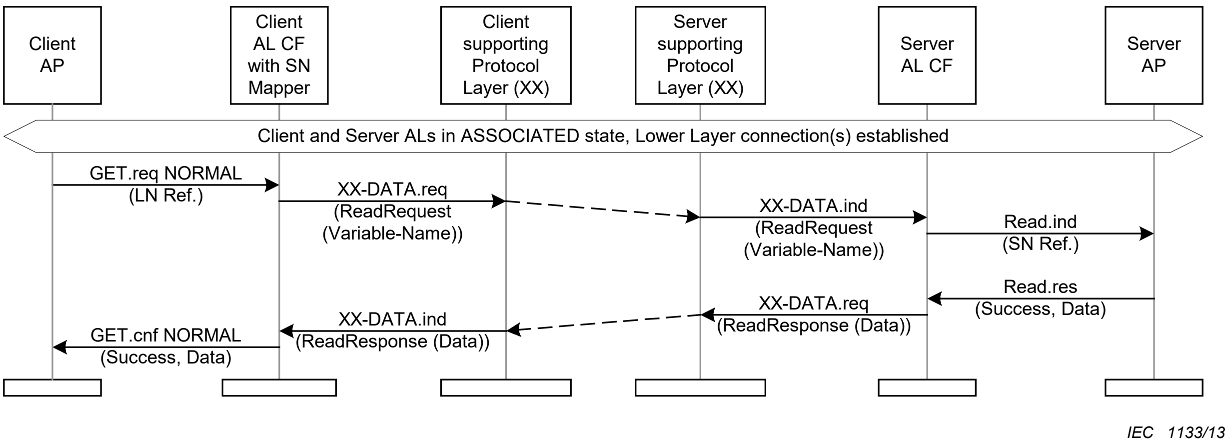


Figure 55 – MSC of the Read service used for reading an attribute

Figure 56 shows the MSC of a Read service used to invoke a single method.

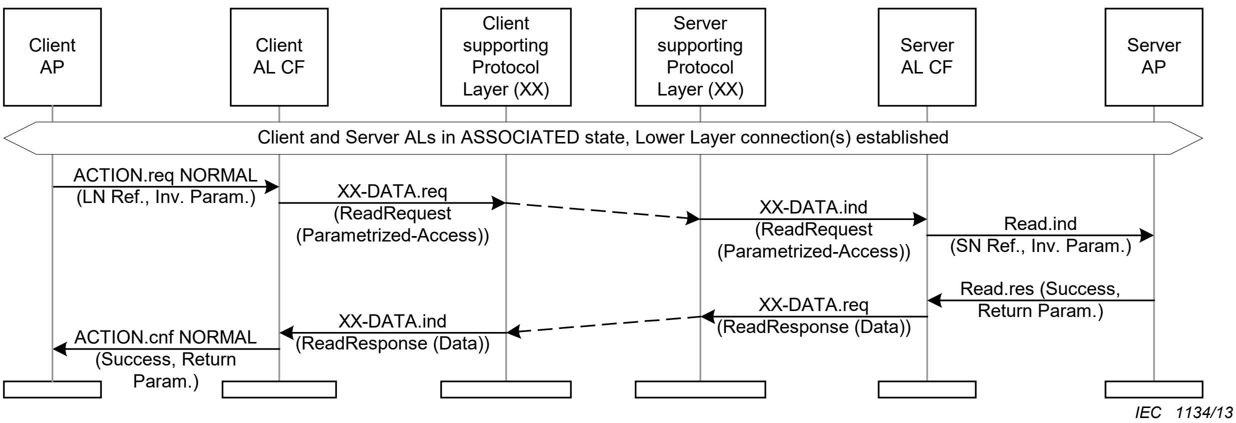
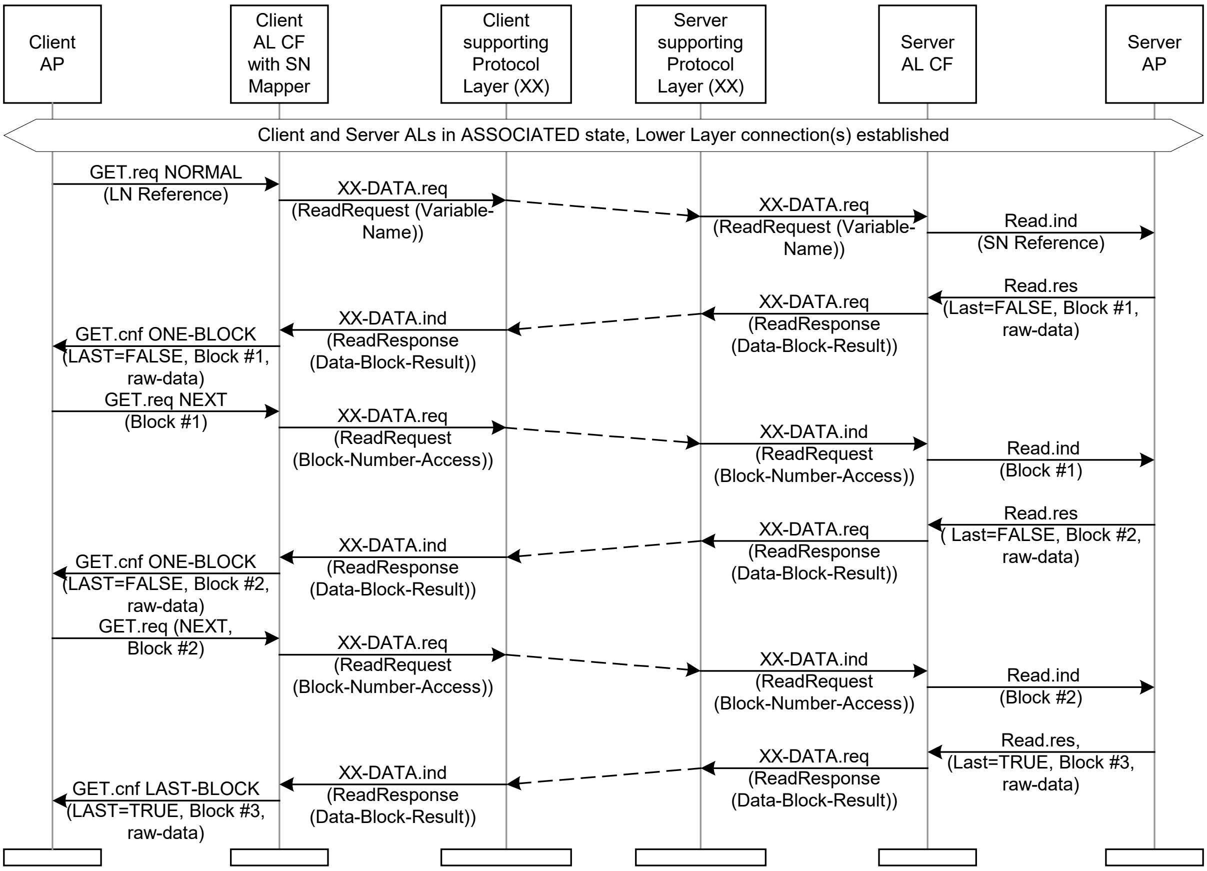


Figure 56 – MSC of the Read service used for invoking a method

Figure 57 shows the MSC of a Read service for reading a single attribute, with the result returned in three blocks using the service-specific block transfer mechanism.

Alternatively, the general block transfer mechanism can be used.



IEC 1135/13

Figure 57 – MSC of the Read Service used for reading an attribute, with block transfer

The process of preparing and transporting the long data is essentially the same as in the case of the GET and ACTION services.

- if the Read service is used to read the value of (a) COSEM object attribute(s), the Raw_Data element of the Data_Block_Result construct carries one part of the list of Read_Result(s);
- if the Read service is used to invoke (a) COSEM object method(s) and long method invocation parameters have to be sent, the Raw_Data element of the Read_Data_Block_Access construct carries one part of the method reference(s) and method invocation parameter(s). If long method invocation responses are returned, the Raw_Data element of the Data_Block_Result construct carries one part of the method invocation response(s).

If an error occurs, the server should return a Read.response service primitive with Data_Access_Error carrying appropriate diagnostic information; for example data-block-number-invalid.

7.3.10 Protocol for the Write service

As explained in 6.15, the Write service is used when the server uses SN referencing, either to write (a) COSEM object attribute(s), or to invoke (a) method(s) when no return parameters are expected:

- in the first case, the SET.request service primitives are mapped to Write.request primitives and the Write.confirm primitives to SET.confirm primitives. The mapping and the corresponding SN APDUs are shown in Table 73;

- in the second case, the ACTION.request service primitives are mapped to Write.request primitives and the Write.response primitives to ACTION.confirm primitives. The mapping and the corresponding SN APDUs are shown in Table 74.

Table 73 – Mapping between the SET and the Write services

From SET.request of type	To Write.request	SN APDU
NORMAL	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name I Parameterized_Access;	WriteRequest::= (variable-name I parameterized-access, data)
FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Data = raw-data, carrying the encoded form of the attribute reference(s) and write data.	WriteRequest::= (write-data-block-access, data)
ONE-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = next number, Data = as above	WriteRequest::= (write-data-block-access, data)
LAST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = TRUE, Block_Number = next number, Data = as above	WriteRequest::= (write-data-block-access, data)
WITH-LIST	Write.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name I Parameterized_Access;	WriteRequest::= ({variable-name I parameterized-access}, {data})
FIRST-BLOCK-WITH-LIST	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1 Data =as above	WriteRequest::= (write-data-block-access, data)

Table 73 *(continued)*

To SET.confirm of type	SN APDU	From Write.response
NORMAL	WriteResponse::= (success I data-access-error)	Write.response (Write_Result) Write_Result = Success I Data_Access_Error;
ACK-BLOCK	WriteResponse::= (block-number)	Write.response (Write_Result) Write_Result = Block_Number;
LAST-BLOCK	WriteResponse::= (success I data-access-error)	Write.response (Write_Result) Write_Result = Success I Data_Access_Error;
WITH-LIST	WriteResponse::= ({success I data-access-error})	Write.response ({Write_Result}) Write_Result = Success I Data_Access_Error;
LAST-BLOCK-WITH-LIST	WriteResponse::= ({success I data-access-error})	Write.response ({Write_Result}) Write_Result = Success I Data_Access_Error;

Table 74 – Mapping between the ACTION and the Write service

From ACTION.request of type	To Write.request	SN APDU
NORMAL	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name; Data = method invocation parameters or null-data	WriteRequest::= (variable-name, data)
FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Data = raw-data, carrying the encoded form of the method reference(s) and method invocation parameters;	WriteRequest::= (write-data-block-access, data)
ONE-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = next number, Data = as above	WriteRequest::= (write-data-block-access, data)
LAST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = TRUE, Block_Number = next number, Data = as above	WriteRequest::= (write-data-block-access, data)

WITH-LIST	Write.request ({Variable_Access_Specification}, {Data}) Variable_Acces_Specification = Variable_Name; Data = method invocation parameters or null data	WriteRequest::= ({variable-name}, {data})
WITH-LIST-AND-FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Data = as with first block	WriteRequest::= (write-data-block-access, data)
To ACTION.confirm	**SN APDU**	**From Write.response**
NORMAL	WriteResponse::= (success I data-access-error)	Write.response (Write_Result) Write_Result = Success I Data_Access_Error
NEXT	WriteResponse::= (block-number)	Write.response (Block_Number)
To ACTION.confirm	**SN APDU**	**From Write.response**
WITH-LIST	WriteResponse::= ({success I data-access-error})	Write.response ({Write_Result}) Write_Result = Success I Data_Access_Error

Figure 58 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.

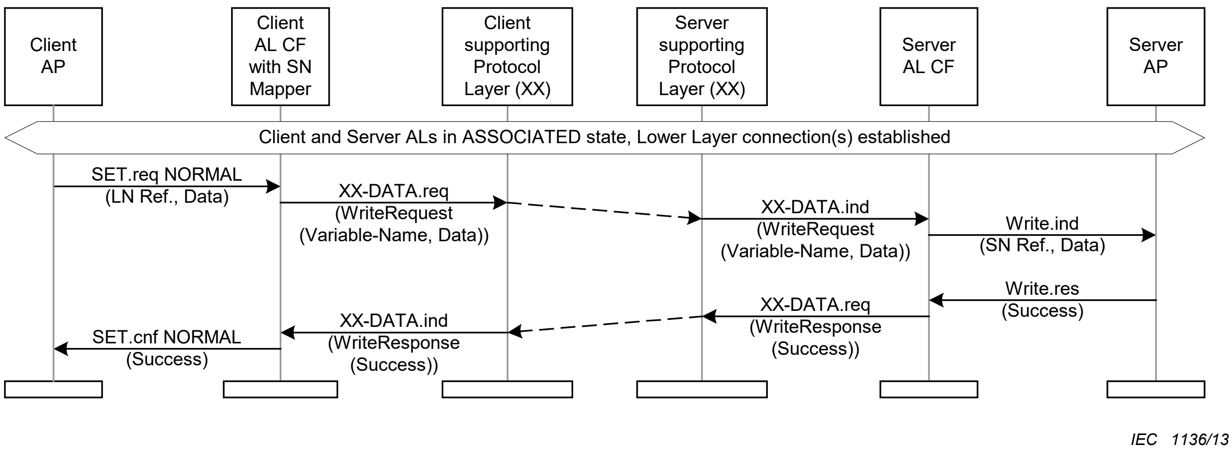


Figure 58 – MSC of the Write service used for writing an attribute

Figure 59shows the MSC of a Write service used to invoke a single method, in the case of success.

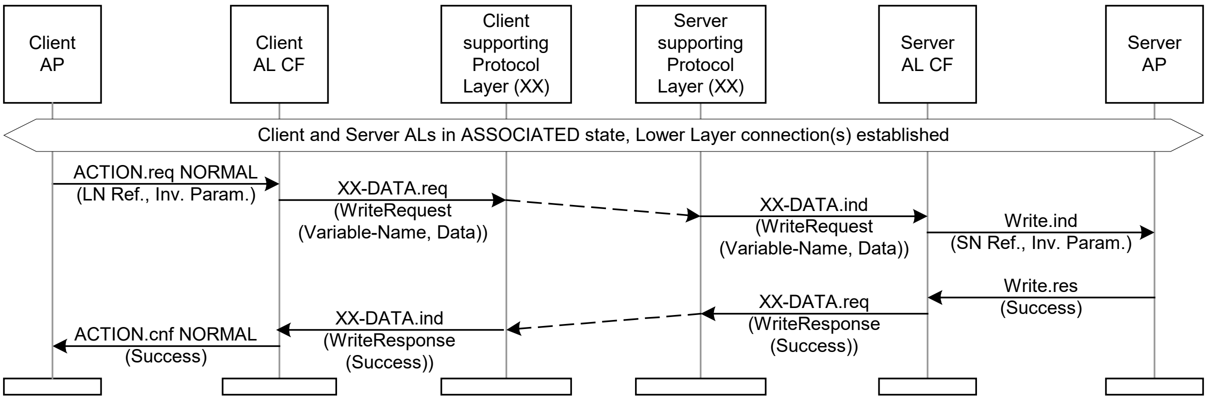


Figure 59 – MSC of the Write service used for invoking a method

Figure 60 shows the MSC of a Write service for writing a single attribute with the result returned in three blocks using the service-specific block transfer mechanism.

Alternatively, the general block transfer mechanism can be used.

The process of preparing and transporting the long data is essentially the same as in the case of the SET and ACTION services:

If an error occurs, the server should return a Write.response service primitive with the Data_Access_Error carrying appropriate diagnostic information; for example data-block-number-invalid.

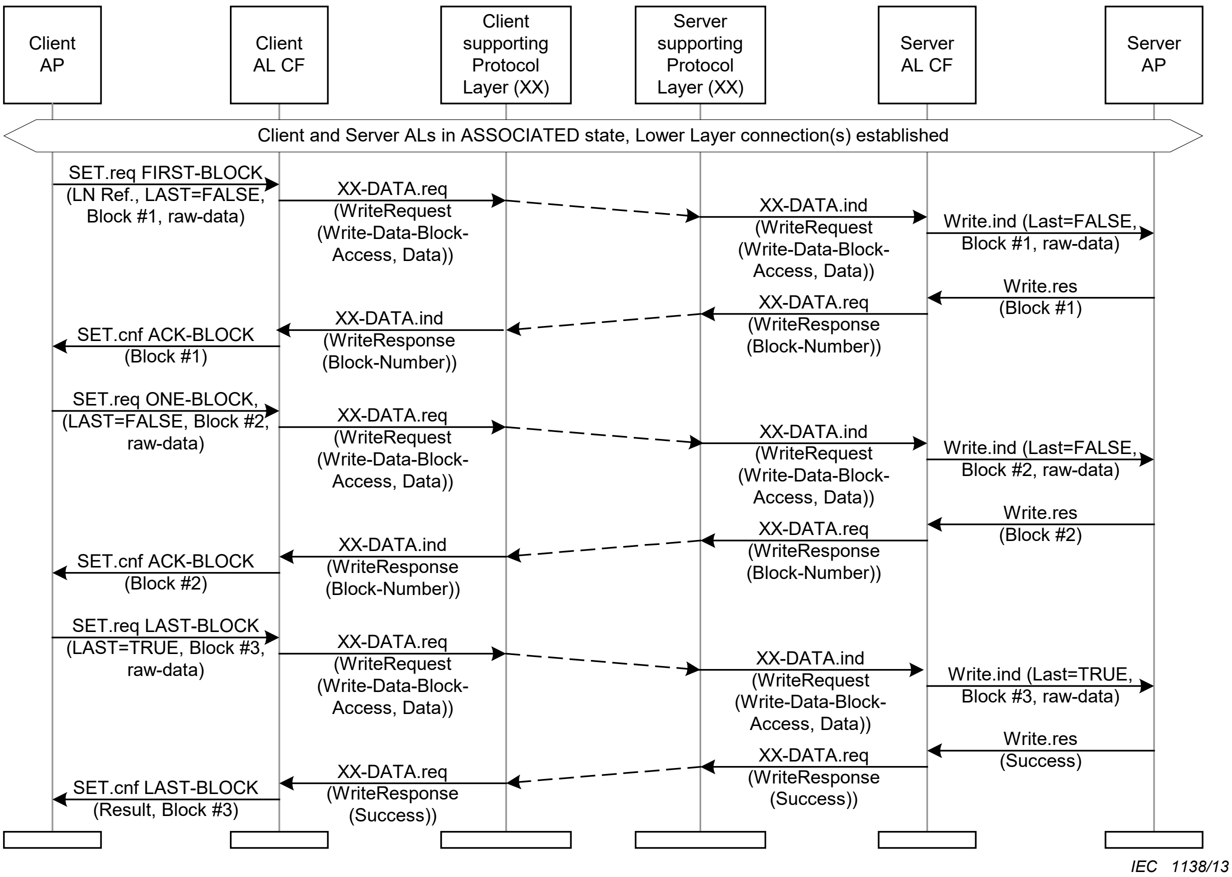


Figure 60 – MSC of the Write Service used for writing an attribute, with block transfer

7.3.11 Protocol for the UnconfirmedWrite service

This service may be invoked only when an AA has already been established. Depending on the communication profile, the APDU corresponding to the request may be transported using the connection-oriented (CO) or connectionless data (CL) services of the supporting protocol layer.

As explained in 6.16, the UnconfirmedWrite service may be used either to write (a) COSEM object attribute(s), or to invoke (a) method(s) when no return parameters are expected:

- in the first case, the SET.request service primitives are mapped to UnconfirmedWrite.request primitives. The mapping and the corresponding SN APDUs are shown in Table 75;
- in the second case, the ACTION.request service primitives are mapped to UnconfirmedWrite.request primitives. The mapping and the corresponding SN APDUs are shown in Table 76.

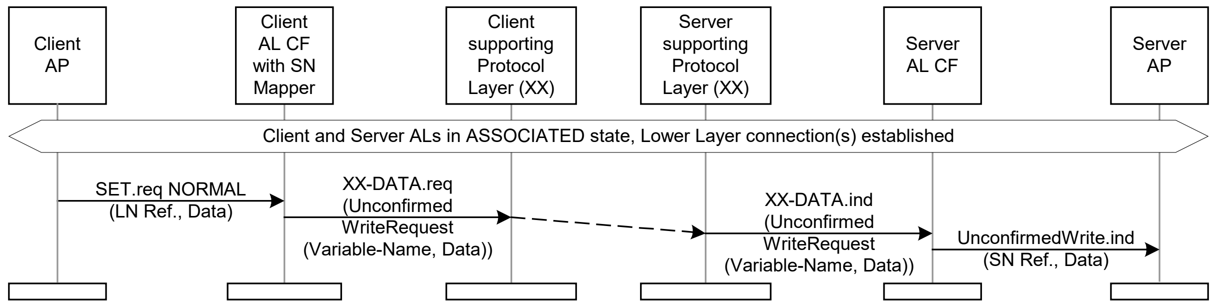
Table 75 – Mapping between the SET and the UnconfirmedWrite services

From SET.request of type	To UnconfirmedWrite.request	SN APDU
NORMAL	UnconfirmedWrite.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name I Parameterized_Access;	UnconfirmedWriteRequest::= (variable-name I parameterized-access, data)
WITH-LIST	UnconfirmedWrite.request ({Variable_Access_Specification}, {Data}) Variable_Acces_Specification = Variable_Name I Parameterized_Access;	UnconfirmedWriteRequest::= ({variable-name I parameterized-access}, {data})

Table 76 – Mapping between the ACTION and the UnconfirmedWrite services

From ACTION.request of type	To UnconfirmedWrite.request	SN APDU
NORMAL	UnconfirmedWrite.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name; Data = method invocation parameters or null data	UnconfirmedWriteRequest::= (variable-name, data)
WITH-LIST	UnconfirmedWrite.request ({Variable_Access_Specification}, {Data}) Variable_Acces_Specification = Variable_Name; Data = as above	UnconfirmedWriteRequest::= ({variable-name}, {data})

Figure 58 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.



IEC 1139/13

Figure 61 – MSC of the Unconfirmed Write service used for writing an attribute

When the service parameters are long, the general block transfer mechanism can be used.

7.3.12 Protocol for the InformationReport service

The protocol for the InformationReport service, specified in 6.17, is essentially the same as that of the EventNotification service, see 7.3.8.

As, unlike the EventNotification service, the InformationReport service does not contain the optional Application_Addresses parameter, the information report is always sent by the Server Management Logical Device to the Client Management AP.

Upon invocation of the InformationReport.request service, the server AP builds an InformationReportRequest APDU. This APDU is sent from the SAP of the management logical device to the SAP of the client management device, using data services of the lower layers, in a non-solicited manner, at the first available opportunity.

The possibilities to send out this APDU depend on the communication profile and the connection status of the lower layers. Therefore, the protocol of the InformationReport service is further discussed in Annex A.

The InformationReport service may carry several attribute names and their contents. On the other hand, the EventNotification service specified in 6.11 contains only one attribute reference. Therefore, when the InformationReportRequest APDU contains more than one attribute, it shall be mapped to several EventNotification.ind services, as shown in Table 77.

Table 77 – Mapping between the EventNotification and InformationReport services

EventNotification.ind (one or more)	InformationReport.ind
Time (optional)	Current-time (optional)
COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id	Variable_Name {Variable_Name}
Attribute_Value	Data {Data}

7.3.13 Protocol of general block transfer mechanism

The general block transfer (GBT) mechanism can be used to carry any xDLMS service primitive when the service parameters are long i.e. their encoded form is longer than the Max Receive PDU Size negotiated. In this case, the AL uses one or more General-Block-Transfer (GBT) xDLMS APDUs to transport such long messages.

The service primitive invocations may be complete including all the service parameters or partial including only one part of the service parameters. Using complete or partial service invocations is left to the implementation.

Following the reception of a service .request / .response service primitive from the AP, the AL:

- builds the APDU that carries the service primitive;
- when ciphering is required it applies the protection as required by the Security_Options and builds the appropriate ciphered APDU;
- when the resulting APDU is longer than the negotiated max APDU size, then the AL uses the GBT mechanism to send the complete message in several GBT APDUs.

However, there is no direct relationship between partial invocations and the GBT APDUs sent. The AL may apply the protection using complete or partial service invocations.

Following the reception of GBT APDUs from a remote party, the AL:

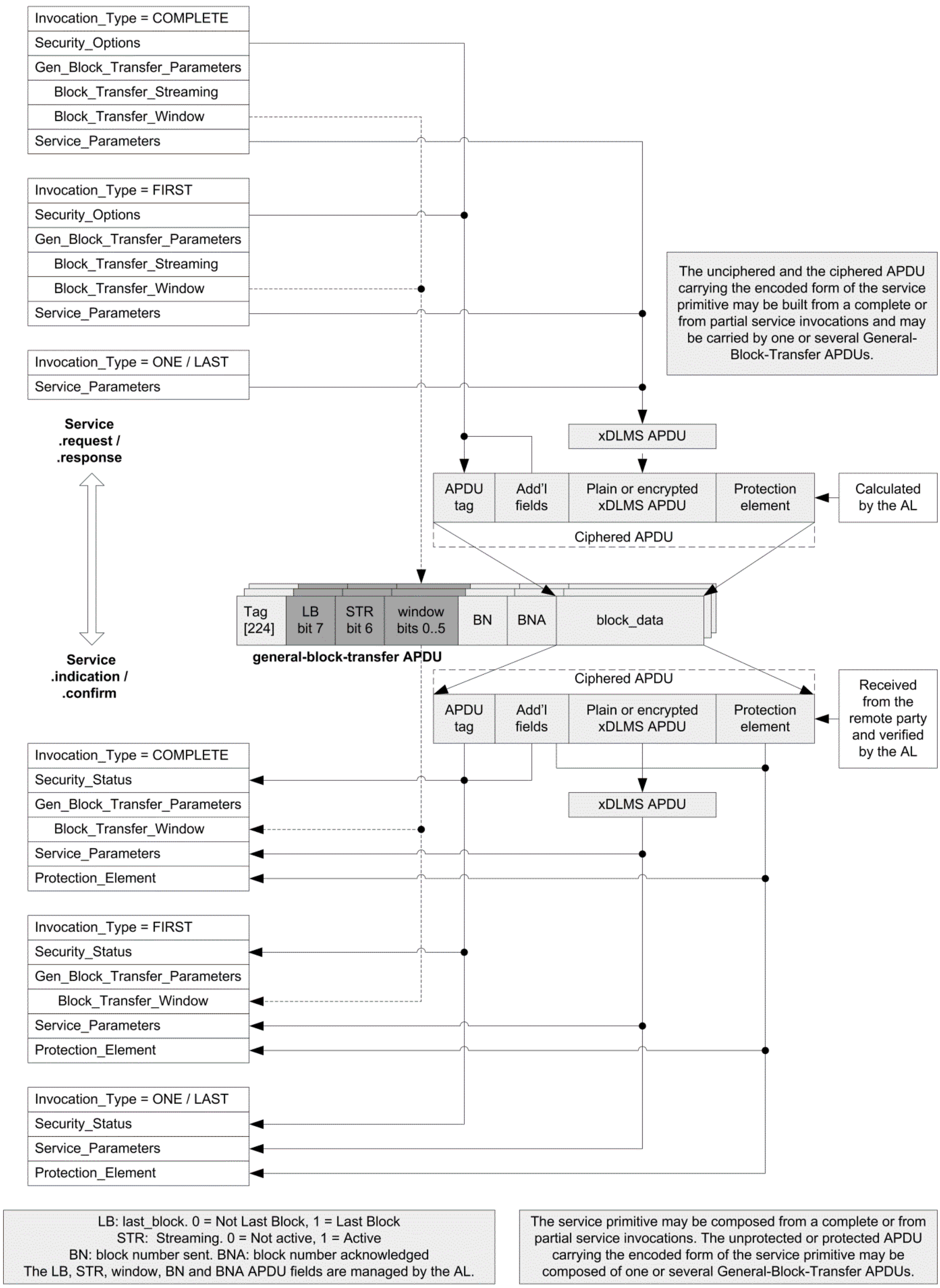
- assembles the block-data fields of the GBT APDUs received together;
- when the resulting complete APDU is ciphered, it checks and removes the protection;
- it invokes the appropriate service primitive, passing the additional Security_Status, the General_Block_Transfer_Parameters and the Protection_Element.

However, there is no direct relationship between the GBT APDUs received and the partial service invocations. The AL may verify and remove the protection processing the GBT APDUs or processing the complete, assembled APDU.

See also Figure 36.

A message exchange may be started without or with using GBT. However, if one party sends a request or a response using GBT, the other party shall follow. The parties continue then using GBT until the end, i.e. until the complete response will have been received.

Streaming of blocks is managed by the AL taking into account the GBT parameters passed from the local AP to the AL – see 6.5 – and the fields of GBT APDUs – see Figure 62 – received from the remote AL.



NOTE Applying and checking/removing cryptographic protection on APDUs is independent from the GBT process. It is included here for completeness.

Figure 62 – Partial service invocations and GBT APDUs

The various service invocation types – COMPLETE, FIRST-PART, ONE-PART or LAST-PART – and the relationship between these invocations, the service parameters and the fields of the ciphered APDUs and the General-Block-Transfer (GBT) APDUs are shown in Figure 59.

The Block_Transfer_Streaming (BTS) parameter is passed by the AP to the AL to indicate that the AL can send blocks in streams, i.e. without waiting for a confirmation of each block received by the remote party. This parameter is not included in the APDU.

The Block_Transfer_Window (BTW) parameter indicates the size of the streaming window supported, i.e. the maximum number of blocks that can be received. The Block_Transfer_Window parameter of the other party may be known *a priori* by the parties. However, the window size is managed by the AL: it can use a lower value, for example during lost block recovery.

NOTE 1 This relationship is indicated using a dotted line in Figure 59 between the Block_Transfer_Window parameter and the window field of the APDU.

In the case of unconfirmed services the Block_Transfer_Streaming parameter shall be set to FALSE and Block_Transfer_Window shall be set to 0. This indicates to the AL that it shall send the encoded form of the whole service primitive in as many GBT APDUs as needed without waiting for confirmation of the blocks sent.

The use of the fields of the GBT APDU is specified below:

- the last-block (LB) bit indicates if the block is the last one (LB = 1) or not (LB = 0);
- the streaming bit indicates if streaming is in progress (STR = 1) or finished (STR = 0). When streaming is finished, the remote party shall confirm the blocks received. When the Block_Transfer_Streaming parameter has been set to FALSE, the streaming bit shall be also set to 0;
- the window field indicates the number of blocks that can be received by the party sending the APDU. Its maximum value is equal to the Block_Transfer_Window parameter passed by the AP to the AL. Note, that the AL may use a lower value during lost block recovery. In the case when the GBT APDUs carry an unconfirmed service (BTS = FALSE, BTW = 0; see above), the value of the window shall be 0 indicating that no GBT APDUs shall be confirmed (and hence no lost blocks can be recovered);
- the block-number (BN) field indicates the number of the block sent. The first block sent shall have block-number = 1. Block-number shall be increased with each GBT APDU sent, even if block-data (BD) is empty. However, during lost block recovery a block number may be repeated;
- the block-number-acknowledged (BNA) field indicates the number of the block acknowledged. If no blocks have been lost, it shall be equal to the number of the last block received. However, if one or more blocks are lost, it shall be equal to the number of the block up to which no blocks are missing;
- the block-data (BD) field carries one part of the xDLMS APDU that is sent using the GBT mechanism.

If a party has no blocks to send, then the last-block bit of the APDU shall be set to 1 and the streaming bit shall be set to 0.

The protocol of the GBT mechanism is further explained with the help of Figure 60, Figure 61, Figure 62, Figure 63, Figure 64, Figure 65 and Figure 66. In these examples, it is assumed that both parties support GBT and six blocks are required to transfer the complete response or request (except in the DataNotification example, where four blocks are required).

NOTE 2 In these examples the service specific block transfer mechanism is not used.

Abbreviations used on the Figures:

- BTS: Block_Transfer_Streaming;
- BTW: Block_Transfer_Window;
- GBT: General-Block-Transfer APDU;
- LB: last-block;
- STR: Streaming;
- BN: block-number;
- BNA: block-number-acknowledged;
- BD (APDU): block-data containing one block of the APDU.

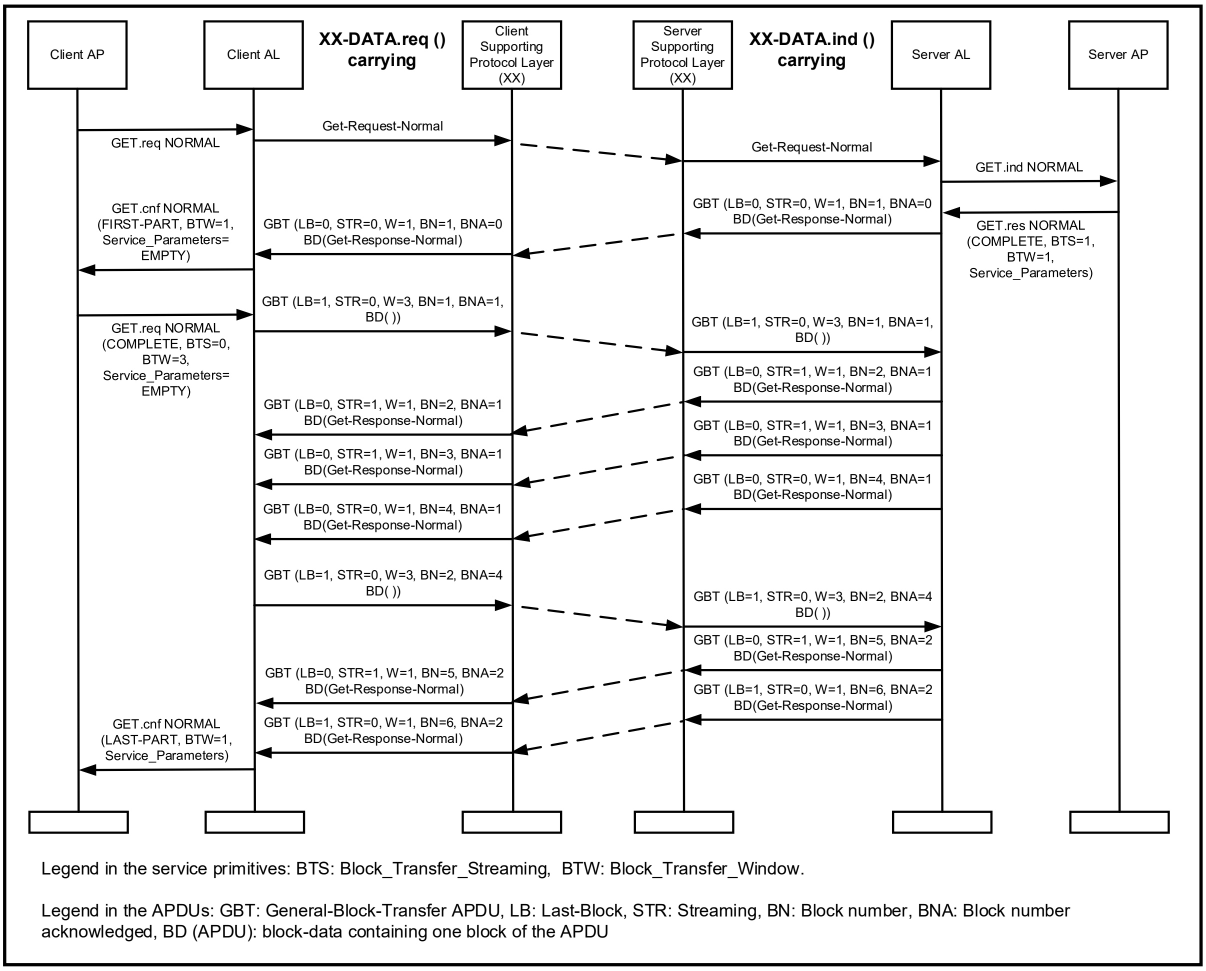


Figure 63 – GET service with GBT, switching to streaming

Figure 63 shows a GET service using GBT. After receiving the first GBT APDU, the client informs the server that it supports streaming. The server switches then to streaming. The process is as follows:

- the client AP invokes a GET.request NORMAL service primitive, without additional service parameters. The client AL sends the request in a Get-Request-Normal APDU;
- the GET.response service parameters are long so the server AP invokes a GET.response NORMAL service primitive with additional service parameters: Invocation_Type = COMPLETE, BTS = 1, BTW = 1 meaning that the server allows sending block streams, but it does not accept block streams from the client. The server AL sends a GBT APDU, containing the first block of the response;

- the client AL invokes a GET.confirm NORMAL service primitive, Invocation_Type = FIRST-PART, BTW = 1. The Service_Parameters are empty. This informs the AP that the response from the server is going to be long;
- the client AP invokes a GET.request NORMAL service primitive, with Invocation_Type = COMPLETE, BTS = 0, BTW = 3, to advertise its capabilities to receive block streams. Note that the Service_Parameters are empty, as these have already been passed in the first GET.request NORMAL service invocation. The client AL sends a GBT APDU. The last-block bit in the APDU is set to 1 and the streaming bit is set to 0 since the client has no blocks to send;
- the server sends then the 2nd (STR = 1, BN = 2), 3rd (STR = 1, BN = 3) and 4th (STR = 0, BN = 4) blocks;
- the client AL sends a GBT APDU to confirm the reception of the 2nd, 3rd and 4th block (LB = 1, STR = 0, W = 3, BN = 2, BNA = 4);
- the server AL sends the 5th (STR =1, BN = 5) and the 6th, last block (LB = 1, STR = 0, BN = 6);

NOTE 2 The last block sent by the server is not confirmed. However, if it is lost, it can be recovered. See Figure 66.

- the client AL invokes a GET.confirm NORMAL service primitive with Invocation_Type = LAST-PART. The service parameters include the complete response to the GET.request.

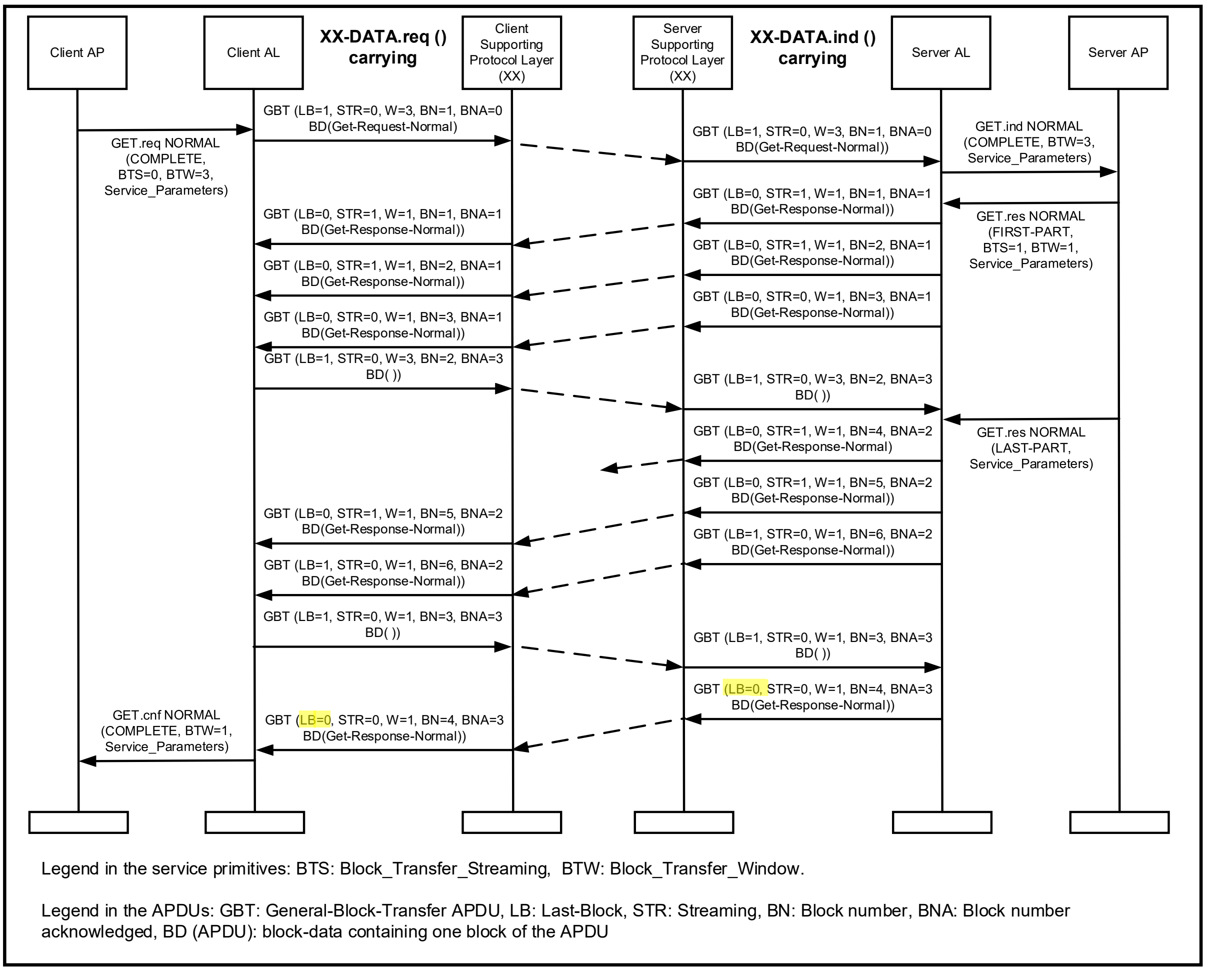
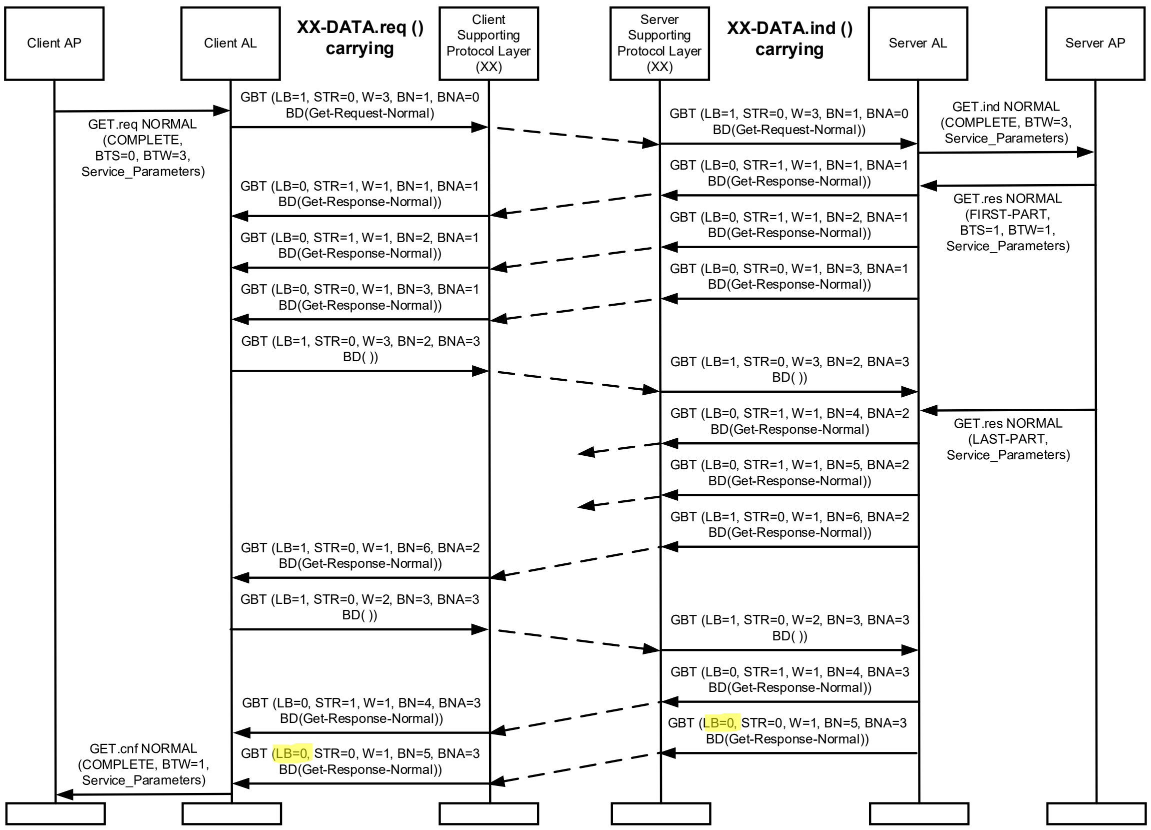


Figure 64 – GET service with partial invocations, GBT and streaming, recovery of 4th block sent in the 2nd stream

Figure 64 shows an example of a GET service using GBT, with partial service invocations on the server side and streaming. The client advertises its streaming capabilities in the first request (BTW = 3; BTW > 1 means that block streams can be received). In this example, the 4^{th} block, sent in the second stream by the server is lost but it is recovered using the following process:

- the client AP invokes a GET.request NORMAL service primitive, with Invocation_Type = COMPLETE, BTS = 0, BTW = 3. The client AL sends a GBT APDU with STR = 0, Window = 3. The server AL invokes the GET.indication NORMAL service primitive with Invocation_Type = COMPLETE, BTW = 3;
- the server AP invokes a GET.response NORMAL service primitive with Invocation_Type = FIRST-PART, BTS = 1, BTW = 1. Service_Parameters include the first part of the response. The server AL sends the 1^{st}, 2^{nd} and 3^{rd} block;
- the client AL sends a GBT APDU to confirm the reception of the three blocks. The server AP invokes a GET.response NORMAL service primitive with Invocation_Type = LAST-PART. Service_Parameters include the second, last part of the response (which in this case is the last part). The server AL sends the 4^{th}, 5^{th} and 6^{th} block (LB = 1, STR = 0, BN = 6). However, the 4^{th} block is lost;
- the client AL indicates that the 4^{th} block has not been received by sending a GBT APDU confirming the reception of the 3^{rd} block (STR = 0, window = 1, BNA = 3). Notice that the client AL reduces the window size to 1 to indicate that only one block has to be re-sent;
- the server sends the 4^{th} block again (LB = 0, STR = 0, BN = 4, BNA = 3);
- now that the client has received now all of the blocks, it invokes a GET.confirm NORMAL service primitive with Invocation_Type = COMPLETE, BTW = 1. The Service_Parameters of this invocation contain the complete response to the GET.request.

Figure 65 shows a scenario which is essentially the same as in Figure 120 except that the 4^{th} and 5^{th} blocks are lost and recovered.



Legend in the service primitives: BTS: Block_Transfer_Streaming, BTW: Block_Transfer_Window.

Legend in the APDUs: GBT: General-Block-Transfer APDU, LB: Last-Block, STR: Streaming, BN: Block number, BNA: Block number acknowledged, BD (APDU): block-data containing one block of the APDU

Figure 65 – GET service with partial invocations, GBT and streaming, recovery of 4th and 5th block Figure title

The process is as follows:

- the client receives the 6th block (LB = 1, STR = 0, BN = 6, BNA = 2);
- the client indicates that the 4th and the 5th blocks have been lost, by sending a GBT APDU with W = 2, BNA = 3, which means that no blocks are missing up to the 3rd block but two blocks have been lost and that the server can send these two using streaming;
- the lost (ie not acknowledged) blocks are then sent by the server. These are the 4th (LB = 0, STR = 1, BN = 4) and 5th (LB = 0, STR = 0, BN = 5). (Note that LB = 0 in the 5th block.) Although it is the last block of the two re-sent blocks, it is not the last block of the original whole message, that was block 6. The original value of LB is preserved during the recovery process.);
- now that the client has received all of the blocks, it invokes a GET.confirm NORMAL service primitive with Invocation_Type = COMPLETE, BTW = 1. The Service_Parameters include the parameters of the complete response to the GET.request.

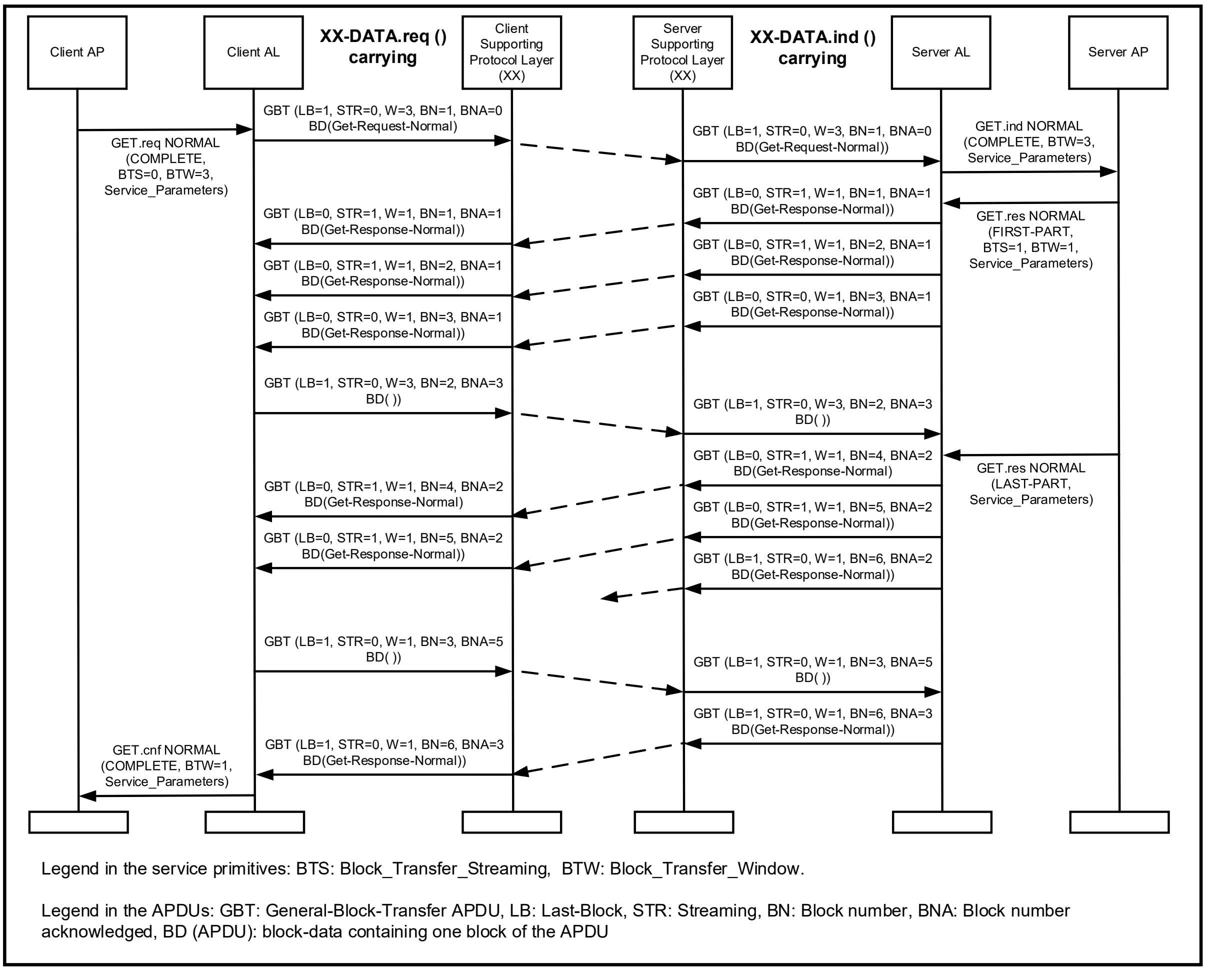


Figure 66 – GET service with partial invocations, GBT and streaming, recovery of last block

Figure 66 shows a scenario when the last block sent in the second stream is lost and is recovered. The process is as follows:

- the client receives the 5th block carried by a GBT APDU (LB = 0, STR = 1, BN = 5);
- as this is not the last block, client waits for a period of time which is implementation specific. If the last block has not been received after this time, it sends a GBT APDU (LB = 1, STR = 0, BN = 3, BNA = 5);
- the server then sends the lost (not confirmed) 6th block carried by a GBT APDU (LB = 1, STR = 0, W = 1, BN = 6 and BNA = 3);
- when the client receives this APDU, it invokes a GET.confirm NORMAL service primitive with Invocation_Type = COMPLETE, BTW = 1. The Service_Parameters include the parameters of the complete response to the GET.request.

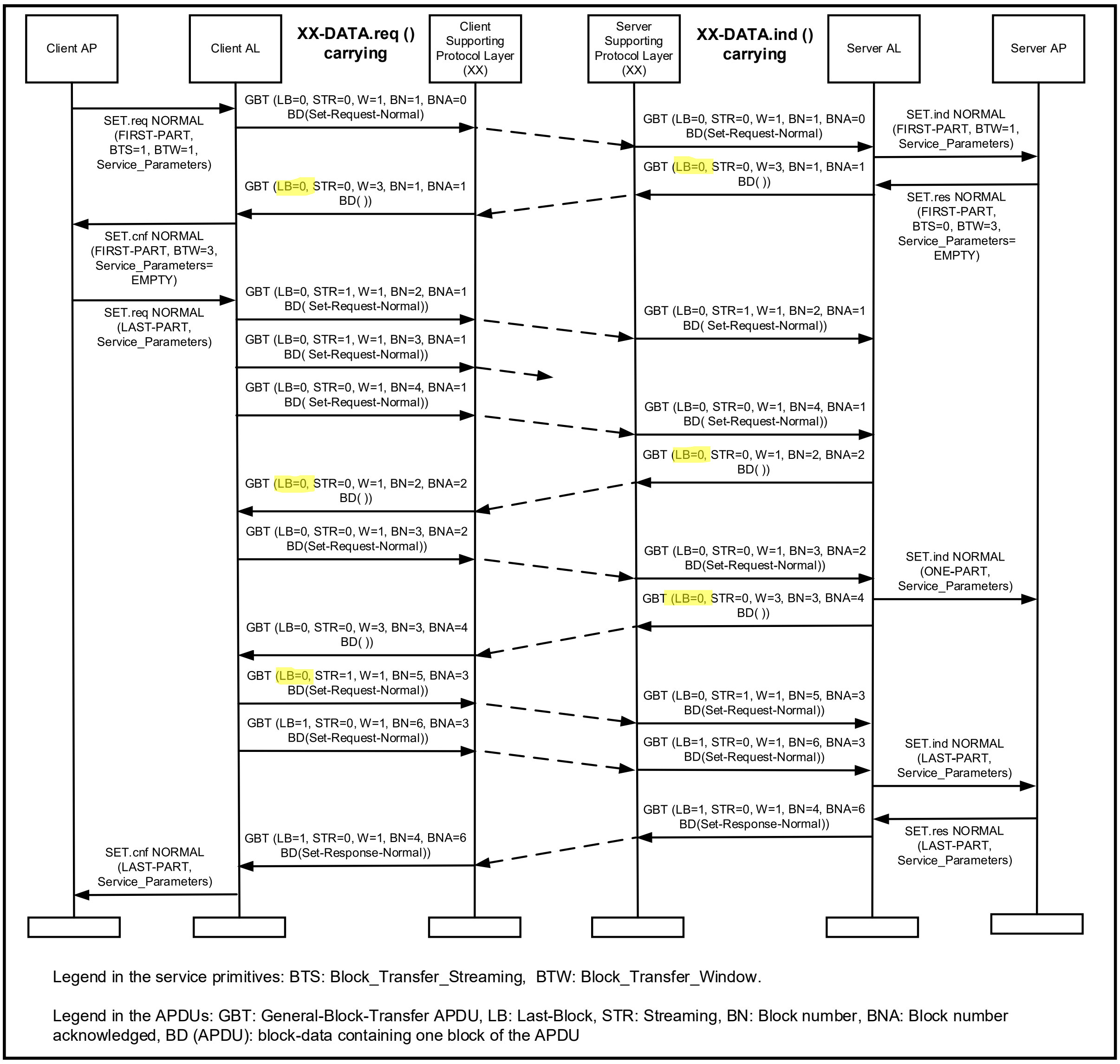
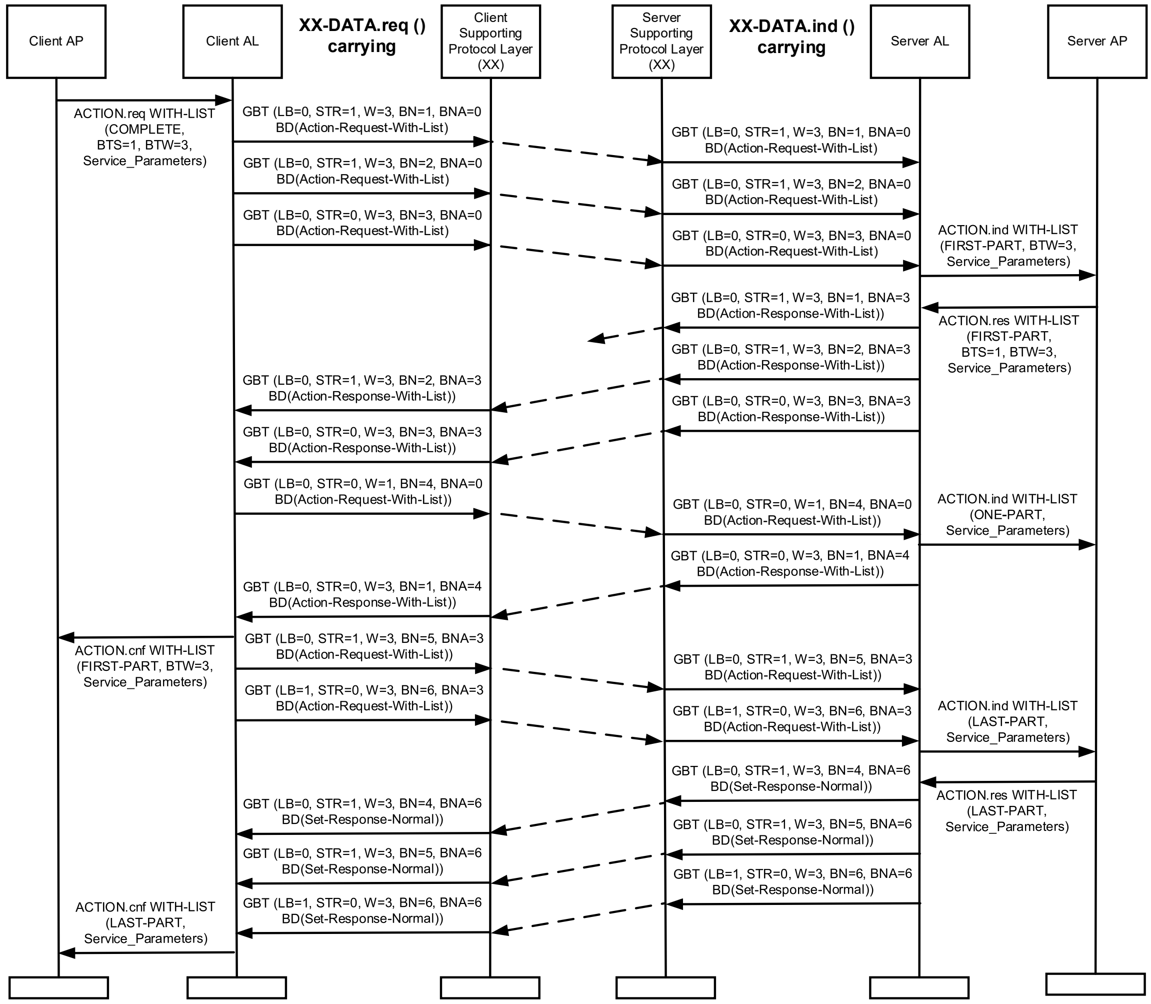


Figure 67 – SET service with GBT, with server not supporting streaming, recovery of 3rd block

Figure 67 shows a SET service with GBT and streaming. In this example, the 3rd block sent by the client is lost and recovered. The process is as follows:

- the client AP invokes a SET.request NORMAL service primitive with Invocation_Type = FIRST-PART, BTS = 1, BTW = 1. The Service_Parameters include the first part of the SET.request. The client AL only sends the first block because it does not know the size of the streaming window supported by the server;
- the server AL invokes a SET.indication NORMAL service primitive with Invocation_Type = FIRST-PART, BTW = 1. The Service_Parameters include the first part of the parameters of the SET.request;
- the server AP responds with a SET.response NORMAL service primitive with Invocation_Type = FIRST-PART, BTS = 0, BTW = 3. The Service_Parameters are empty. The server AL sends a GBT APDU with LB = 0, STR = 0, Window = 3; block-data is empty. This informs the client that the server can receive block streams and the window size = 3. It therefore sends the 2nd, 3rd and 4th block in a stream. However, the 3rd block is lost;

- the server indicates that block 3 is lost by confirming the reception of the 2nd block and reduces the window size to 1 (LB = 0, STR = 0, W = 1, BN = 2, BNA = 2). The client sends then the 3rd block again (LB = 0, STR = 0, BN = 3, BNA = 2);
- the server AL invokes a SET.indication NORMAL service primitive with Invocation_Type = ONE-PART. The server AL confirms the reception of the blocks up to the 4th block (LB = 0, STR = 0, W = 3, BNA = 4). Notice that the window size has been increased again to 3;
- the client then sends the 5th and the 6th block using streaming;
- when the server AL receives the 6th block which is the last block, it invokes a SET.indication NORMAL service primitive with Invocation_Type = LAST-PART. Service_Parameters include the last part of the parameters of the SET.request;
- the server AP invokes a SET.response NORMAL service primitive with Invocation_Type = LAST-PART, with the Service_Parameters containing the result of the set operation(s). This is sent by the server AL in a GBT APDU (LB = 1, BN = 4, BNA = 6). The client AL invokes the SET.confirm NORMAL service primitive with Invocation_Type = LAST-PART. Service_Parameters include the result of the set operations.



Legend in the service primitives: BTS: Block_Transfer_Streaming, BTW: Block_Transfer_Window.

Legend in the APDUs: GBT: General-Block-Transfer APDU, LB: Last-Block, STR: Streaming, BN: Block number, BNA: Block number acknowledged, BD (APDU): block-data containing one block of the APDU

Figure 68 – ACTION-WITH-LIST service with bi-directional GBT and block recovery

Figure 68 shows an ACTION-WITH-LIST service with partial service invocations, bidirectional block transfer and streaming. Each party already knows that the other party supports streaming with window size = 3. In this example, the first block sent by the server is lost and recovered. The process is as follows:

- the client invokes and ACTION.request of type WITH-LIST service primitive with Invocation_Type = COMPLETE, BTS = 1, BTW = 3. The client AL sends the first three blocks that carry a part of this request to the server. The server AL invokes and ACTION.indication of type WITH-LIST service primitive with Invocation_Type = FIRST-PART, BTW = 3. Service parameters contain the first part of the request;
- the server AP processes this request and has the first part of the response available. It invokes an ACTION.response of type WITH-LIST service primitive with Invocation_Type = FIRST-PART, BTS = 1, BTW = 3. The server AL sends this in three blocks using streaming. However, the 1st block is lost;
- the client AL asks the server to send the lost 1st block again by not confirming any blocks received. It also sends its 4th block (LB = 0, STR = 0, W = 1, BN = 4, BNA = 0). Notice that the client AL has reduced the window size to 1;
- the server AL invokes an ACTION.indication of type WITH-LIST service primitive with INVOCATION_Type = ONE-PART. Service_Parameters contain one part of the request;

- the server sends the lost 1st block and confirms the 4th block received from the client (BN = 1, BNA = 4);
- the client AL invokes an ACTION.confirm of type WITH-LIST service primitive with additional parameters: Invocation_Type = FIRST-PART, BTW = 3. The Service_Parameters include one part of the response from the server;
- the client AL sends the 5th and the 6th,(last) block. The window size is increased again to 3;
- the server AL invokes an ACTION.indication of type WITH-LIST service primitive with Invocation_Type = LAST-PART and with Service_Parameters containing the last part of the ACTION.request;
- the server AP processes this and invokes an ACTION.response of type WITH-LIST service primitive with Invocation_Type = LAST-PART; the Service_Parameters contain the remaining part of the response. This is sent to client in three blocks using streaming;
- the client AL invokes an ACTION.confirm of type WITH-LIST service primitive with Invocation_Type = LAST-PART. The Service_Parameters include the last part of the response from the server.

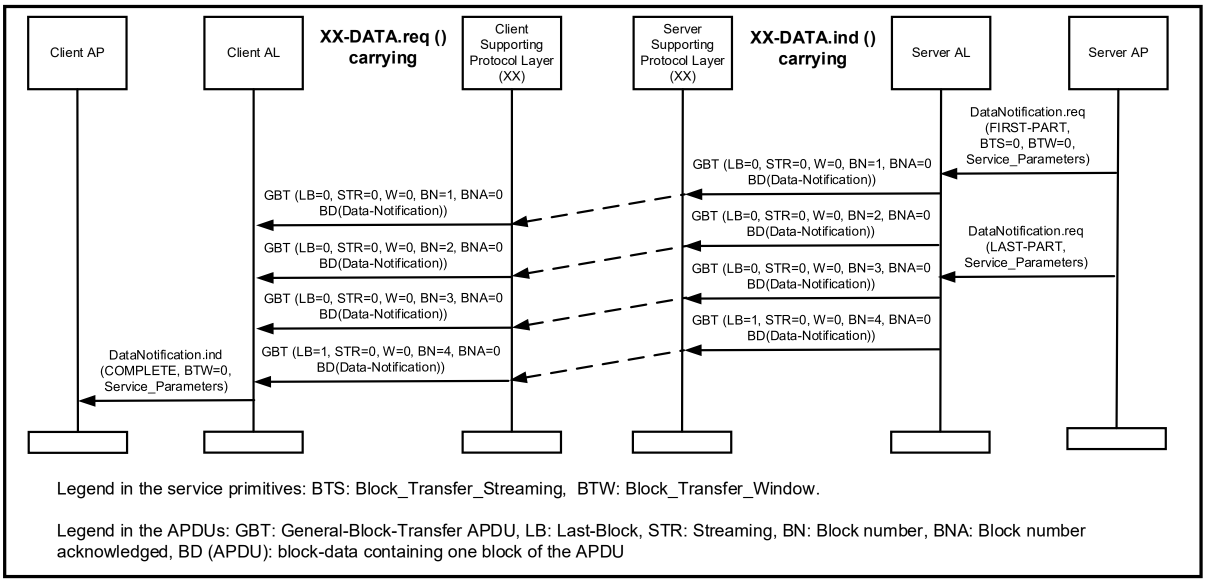


Figure 69 – DataNotification service with GBT with partial invocation

Figure 69 shows a DataNotification service with GBT, with partial service invocations on the server side. The process is the following:

- the server AP invokes a DataNotification.request service primitive with Invocation_Type = FIRST-PART, BTS = 0, BTW = 0. The Service_Parameters include one part of the DataNotification.request;
- the server AL sends the GBT APDUs to the client. The reception of the blocks is not confirmed, block recovery is not available;
- when the client AL receives the last block, it assembles the block-data together and invokes a DataNotification.indication service primitive with Invocation_Type = COMPLETE, BTW = 0. The Service_Parameters include the complete DataNotification service parameters.

Aborting the GBT process

The client or the server may want to abort the GBT process. To do so, it shall send a GBT APDU with LB = 1, STR = 0, BN = 0 and BNA = 0. The block transfer process shall also be aborted if a party confirms the reception of a block not yet sent by the other party.

It is not possible to abort GBT with DataNotification.

8 Abstract syntax of ACSE and COSEM APDUs

The abstract syntax of COSEM APDUs is specified in this clause using ASN.1. See ISO/IEC 8824-1:2008.

```
COSEMpdu DEFINITIONS::= BEGIN

ACSE-APDU::= CHOICE
{
    aarq                                AARQ-apdu,
    aare                                AARE-apdu,
    rlrq                                RLRQ-apdu,           -- OPTIONAL
    rlre                                RLRE-apdu            -- OPTIONAL
}

XDLMS-APDU::= CHOICE
{
-- standardised xDLMS pdus used in DLMS/COSEM

-- with no ciphering

    initiateRequest                     [1]  IMPLICIT       InitiateRequest,
    readRequest                         [5]  IMPLICIT       ReadRequest,
    writeRequest                        [6]  IMPLICIT       WriteRequest,

    initiateResponse                    [8]  IMPLICIT       InitiateResponse,
    readResponse                        [12] IMPLICIT       ReadResponse,
    writeResponse                       [13] IMPLICIT       WriteResponse,

    confirmedServiceError               [14]                ConfirmedServiceError,

-- data-notification

```

```
    data-notification                  [15] IMPLICIT       Data-Notification,
    data-notification-confirm          [16] IMPLICIT       Data-Notification-Confirm,
    unconfirmedWriteRequest            [22] IMPLICIT       UnconfirmedWriteRequest,
    informationReportRequest           [24] IMPLICIT       InformationReportRequest,

-- The APDU tag of each ciphered xDLMS APDU indicates the type of the unciphered APDU and whether
-- global or dedicated key is used. The type of the key is carried by the security header, and after
-- removing the encryption and/or verifying the authentication tag, the original APDU with its APDU
-- TAG is restored. Therefore, the APDU tags of the ciphered APDUs carry redundant information, but
-- they are retained for consistency.

-- with global ciphering

    glo-initiateRequest                [33]  IMPLICIT      OCTET STRING,
    glo-readRequest                    [37]  IMPLICIT      OCTET STRING,
    glo-writeRequest                   [38]  IMPLICIT      OCTET STRING,

    glo-initiateResponse               [40]  IMPLICIT      OCTET STRING,
    glo-readResponse                   [44]  IMPLICIT      OCTET STRING,
    glo-writeResponse                  [45]  IMPLICIT      OCTET STRING,

    glo-confirmedServiceError          [46]  IMPLICIT      OCTET STRING,

    glo-unconfirmedWriteRequest        [54]  IMPLICIT      OCTET STRING,
    glo-informationReportRequest       [56]  IMPLICIT      OCTET STRING,

-- with dedicated ciphering

-- not used in DLMS/COSEM
    ded-initiateRequest                [65]  IMPLICIT      OCTET STRING,

    ded-readRequest                    [69]  IMPLICIT      OCTET STRING,
```

```
    ded-writeRequest                   [70]  IMPLICIT      OCTET STRING,

-- not used in DLMS/COSEM
    ded-initiateResponse               [72]  IMPLICIT      OCTET STRING,

    ded-readResponse                   [76]  IMPLICIT      OCTET STRING,
    ded-writeResponse                  [77]  IMPLICIT      OCTET STRING,

    ded-confirmedServiceError          [78]  IMPLICIT      OCTET STRING,

    ded-unconfirmedWriteRequest        [86]  IMPLICIT      OCTET STRING,
    ded-informationReportRequest       [88]  IMPLICIT      OCTET STRING,

-- xDLMS APDUs used with LN referencing
-- with no ciphering

    get-request                        [192] IMPLICIT      Get-Request,
    set-request                        [193] IMPLICIT      Set-Request,
    event-notification-request         [194] IMPLICIT      EventNotificationRequest,
    action-request                     [195] IMPLICIT      Action-Request,

    get-response                       [196] IMPLICIT      Get-Response,
    set-response                       [197] IMPLICIT      Set-Response,
    action-response                    [199] IMPLICIT      Action-Response,

-- with global ciphering

    glo-get-request                    [200] IMPLICIT      OCTET STRING,
    glo-set-request                    [201] IMPLICIT      OCTET STRING,
    glo-event-notification-request     [202] IMPLICIT      OCTET STRING,
    glo-action-request                 [203] IMPLICIT      OCTET STRING,
```

```
    glo-get-response                   [204] IMPLICIT      OCTET STRING,
    glo-set-response                   [205] IMPLICIT      OCTET STRING,
    glo-action-response                [207] IMPLICIT      OCTET STRING,

-- with dedicated ciphering

    ded-get-request                    [208] IMPLICIT      OCTET STRING,
    ded-set-request                    [209] IMPLICIT      OCTET STRING,
    ded-event-notification-request     [210] IMPLICIT      OCTET STRING,
    ded-actionRequest                  [211] IMPLICIT      OCTET STRING,

    ded-get-response                   [212] IMPLICIT      OCTET STRING,
    ded-set-response                   [213] IMPLICIT      OCTET STRING,
    ded-action-response                [215] IMPLICIT      OCTET STRING,

-- the exception response pdu

    exception-response                 [216] IMPLICIT      ExceptionResponse,


-- access

    access-request                     [217] IMPLICIT      Access-Request,
    access-response                    [218] IMPLICIT      Access-Response,

-- general APDUs
    general-glo-ciphering              [219] IMPLICIT      General-Glo-Ciphering,
    general-ded-ciphering              [220] IMPLICIT      General-Ded-Ciphering,
    general-ciphering                  [221] IMPLICIT      General-Ciphering,
    general-signing                    [223] IMPLICIT      General-Signing,
    general-block-transfer             [224] IMPLICIT      General-Block-Transfer
```

```
-- The tags 230 and 231 are reserved for DLMS Gateway
-- reserved                            [230]
-- reserved                            [231]
}

AARQ::= [APPLICATION 0] IMPLICIT SEQUENCE
{
-- [APPLICATION 0] == [ 60H ] = [ 96 ]

    protocol-version                   [0] IMPLICIT       BIT STRING {version1 (0)} DEFAULT {version1},
    application-context-name           [1]                Application-context-name,
    called-AP-title                    [2]                AP-title OPTIONAL,
    called-AE-qualifier                [3]                AE-qualifier OPTIONAL,
    called-AP-invocation-id            [4]                AP-invocation-identifier OPTIONAL,
    called-AE-invocation-id            [5]                AE-invocation-identifier OPTIONAL,
    calling-AP-title                   [6]                AP-title OPTIONAL,
    calling-AE-qualifier               [7]                AE-qualifier OPTIONAL,
    calling-AP-invocation-id           [8]                AP-invocation-identifier OPTIONAL,
    calling-AE-invocation-id           [9]                AE-invocation-identifier OPTIONAL,

-- The following field shall not be present if only the kernel is used.
    sender-acse-requirements           [10] IMPLICIT     ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is selected.
    mechanism-name                     [11] IMPLICIT     Mechanism-name OPTIONAL,

-- The following field shall only be present if the authentication functional unit is selected.

    calling-authentication-value       [12] EXPLICIT     Authentication-value OPTIONAL,
    implementation-information         [29] IMPLICIT     Implementation-data OPTIONAL,
    user-information                   [30] EXPLICIT     Association-information OPTIONAL
}
```

```
-- The user-information field shall carry an InitiateRequest APDU encoded in A-XDR, and then
-- encoding the resulting OCTET STRING in BER.

AARE-apdu::= [APPLICATION 1] IMPLICIT SEQUENCE
{
-- [APPLICATION 1] == [ 61H ] = [ 97 ]

    protocol-version                  [0] IMPLICIT        BIT STRING {version1 (0)} DEFAULT {version1},
    application-context-name          [1]                 Application-context-name,
    result                            [2]                 Association-result,
    result-source-diagnostic          [3]                 Associate-source-diagnostic,
    responding-AP-title               [4]                 AP-title OPTIONAL,
    responding-AE-qualifier           [5]                 AE-qualifier OPTIONAL,
    responding-AP-invocation-id       [6]                 AP-invocation-identifier OPTIONAL,
    responding-AE-invocation-id       [7]                 AE-invocation-identifier OPTIONAL,

-- The following field shall not be present if only the kernel is used.
    responder-acse-requirements       [8] IMPLICIT        ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is selected.
    mechanism-name                    [9] IMPLICIT        Mechanism-name OPTIONAL,

-- The following field shall only be present if the authentication functional unit is selected.
    responding-authentication-value   [10] EXPLICIT       Authentication-value OPTIONAL,
    implementation-information        [29] IMPLICIT       Implementation-data OPTIONAL,
    user-information                  [30] EXPLICIT       Association-information OPTIONAL
}

-- The user-information field shall carry either an InitiateResponse (or, when the proposed xDLMS
-- context is not accepted by the server, a confirmedServiceError) APDU encoded in A-XDR, and then
-- encoding the resulting OCTET STRING in BER.
```

```
RLRQ-apdu::= [APPLICATION 2] IMPLICIT SEQUENCE
{
-- [APPLICATION 2] == [ 62H ] = [ 98 ]

    reason                          [0] IMPLICIT       Release-request-reason OPTIONAL,
    user-information                [30] EXPLICIT      Association-information OPTIONAL
}

RLRE-apdu::= [APPLICATION 3] IMPLICIT SEQUENCE
{
-- [APPLICATION 3] == [ 63H ] = [ 99 ]

    reason                          [0] IMPLICIT       Release-response-reason OPTIONAL,
    user-information                [30] EXPLICIT      Association-information OPTIONAL
}

-- The user-information field of the RLRQ / RLRE APDU may carry an InitiateRequest APDU encoded in
-- A-XDR, and then encoding the resulting OCTET STRING in BER, when the AA to be released uses
-- ciphering.

-- types used in the fields of the ACSE APDUs, in the order of their occurrence

Application-context-name::=         OBJECT IDENTIFIER

AP-title::=                         OCTET STRING

AE-qualifier::=                     OCTET STRING

AP-invocation-identifier::=         INTEGER

AE-invocation-identifier::=         INTEGER
```

```
ACSE-requirements::=                  BIT STRING {authentication(0)}

Mechanism-name::=                     OBJECT IDENTIFIER

Authentication-value::= CHOICE
{
    charstring                         [0] IMPLICIT    GraphicString,
    bitstring                          [1] IMPLICIT    BIT STRING
}

Implementation-data::=                GraphicString

Association-information::=            OCTET STRING

Association-result::=                 INTEGER
{
    accepted                           (0),
    rejected-permanent                 (1),
    rejected-transient                 (2)
}

Associate-source-diagnostic::= CHOICE
{
    acse-service-user                  [1] INTEGER
    {
         null                                              (0),
         no-reason-given                                   (1),
         application-context-name-not-supported            (2),
         calling-AP-title-not-recognized                   (3),
         calling-AP-invocation-identifier-not-recognized  (4),
         calling-AE-qualifier-not-recognized               (5),
```

```
            calling-AE-invocation-identifier-not-recognized  (6),
            called-AP-title-not-recognized                   (7),
            called-AP-invocation-identifier-not-recognized   (8),
            called-AE-qualifier-not-recognized               (9),
            called-AE-invocation-identifier-not-recognized   (10),
            authentication-mechanism-name-not-recognised     (11),
            authentication-mechanism-name-required           (12),
            authentication-failure                           (13),
            authentication-required                          (14)
        },
        acse-service-provider              [2] INTEGER
        {
            null                               (0),
            no-reason-given                    (1),
            no-common-acse-version             (2)
        }
    }

    Release-request-reason::= INTEGER
    {
        normal                             (0),
        urgent                             (1),
        user-defined                       (30)
    }

    Release-response-reason::= INTEGER
    {
        normal                             (0),
        not-finished                       (1),
        user-defined                       (30)
    }
```

```
-- Useful types

Integer8::=                     INTEGER(-128..127)
Integer16::=                    INTEGER(-32768..32767)
Integer32::=                    INTEGER(-2147483648..2147483647)
Integer64::=                    INTEGER(-9223372036854775808..9223372036854775807)
Unsigned8::=                    INTEGER(0..255)
Unsigned16::=                   INTEGER(0..65535)
Unsigned32::=                   INTEGER(0..4294967295)
Unsigned64::=                   INTEGER(0..18446744073709551615)


-- xDLMS APDU-s used during Association establishment

InitiateRequest::= SEQUENCE
{
--  shall not be encoded in DLMS without ciphering
    dedicated-key                     OCTET STRING OPTIONAL,
    response-allowed                  BOOLEAN DEFAULT TRUE,
    proposed-quality-of-service       [0] IMPLICIT Integer8 OPTIONAL,
    proposed-dlms-version-number      Unsigned8,
    proposed-conformance              Conformance, -- Shall be encoded in BER
    client-max-receive-pdu-size       Unsigned16
}

-- In DLMS/COSEM, the quality-of-service parameter is not used. Any value shall be accepted.

-- The Conformance field shall be encoded in BER. See IEC 61334-6 Example 1.

InitiateResponse::= SEQUENCE
{
```

```
    negotiated-quality-of-service      [0] IMPLICIT Integer8 OPTIONAL,
    negotiated-dlms-version-number     Unsigned8,
    negotiated-conformance             Conformance, -- Shall be encoded in BER
    server-max-receive-pdu-size        Unsigned16,
    vaa-name                           ObjectName
}

-- In the case of LN referencing, the value of the vaa-name is 0x0007
-- In the case of SN referencing, the value of the vaa-name is the base name of the
-- Current Association object, 0xFA00

-- Conformance Block

-- SIZE constrained BIT STRING is extension of ASN.1 notation

Conformance::= [APPLICATION 31] IMPLICIT BIT STRING
{
    -- the bit is set when the corresponding service or functionality is available
    reserved-zero                      (0),
    -- The actual list of general protection services depends on the security suite
    general-protection                 (1),
    general-block-transfer             (2),
    read                               (3),
    write                              (4),
    unconfirmed-write                  (5),
    delta-value-encoding               (6),
    reserved-seven                     (7),
    attribute0-supported-with-set      (8),
    priority-mgmt-supported            (9),
    attribute0-supported-with-get      (10),
    block-transfer-with-get-or-read    (11),
    block-transfer-with-set-or-write   (12),
```

```
    block-transfer-with-action        (13),
    multiple-references               (14),
    information-report                (15),
    data-notification                 (16),
    access                            (17),
    parameterized-access              (18),
    get                               (19),
    set                               (20),
    selective-access                  (21),
    event-notification                (22),
    action                            (23)
}

ObjectName::=                             Integer16
-- for named variable objects (short names), the last three bits shall be set to 000;
-- for vaa-name objects, the last three bits shall be set to 111.

-- The Confirmed ServiceError APDU is used only with the InitiateRequest, ReadRequest and
-- WriteRequest APDUs when the request fails, to provide diagnostic information.

ConfirmedServiceError::= CHOICE
{
-- tag 0 is reserved
-- In DLMS/COSEM only initiateError, read and write are relevant

    initiateError                     [1] ServiceError,
    getStatus                         [2] ServiceError,
    getNameList                       [3] ServiceError,
    getVariableAttribute              [4] ServiceError,
    read                              [5] ServiceError,
    write                             [6] ServiceError,
    getDataSetAttribute               [7] ServiceError,
```

```
    getTIAttribute                    [8] ServiceError,
    changeScope                       [9] ServiceError,
    start                             [10] ServiceError,
    stop                              [11] ServiceError,
    resume                            [12] ServiceError,
    makeUsable                        [13] ServiceError,
    initiateLoad                      [14] ServiceError,
    loadSegment                       [15] ServiceError,
    terminateLoad                     [16] ServiceError,
    initiateUpLoad                    [17] ServiceError,
    upLoadSegment                     [18] ServiceError,
    terminateUpLoad                   [19] ServiceError
}

ServiceError ::= CHOICE
{
    application-reference             [0] IMPLICIT ENUMERATED
    {
    -- DLMS provider only
        other                              (0),
        time-elapsed                       (1),  -- time out since request sent
        application-unreachable            (2),  -- peer AEi not reachable
        application-reference-invalid      (3),  -- addressing trouble
        application-context-unsupported    (4),  -- application-context incompatibility
        provider-communication-error       (5),  -- error at the local or distant equipment
        deciphering-error                  (6)   -- error detected by the deciphering function
    },

    hardware-resource                 [1] IMPLICIT ENUMERATED
    {
    -- VDE hardware troubles
        other                              (0),
```

```
        memory-unavailable                  (1),
        processor-resource-unavailable      (2),
        mass-storage-unavailable            (3),
        other-resource-unavailable          (4)
    },

    vde-state-error                    [2] IMPLICIT ENUMERATED
    {
    -- Error source description
        other                               (0),
        no-dlms-context                     (1),
        loading-data-set                    (2),
        status-nochange                     (3),
        status-inoperable                   (4)
    },

    service                            [3] IMPLICIT ENUMERATED
    {
    -- service handling troubles
        other                               (0),
        pdu-size                            (1),  -- pdu too long
        service-unsupported                 (2)   -- as defined in the conformance block
    },

    definition                         [4] IMPLICIT ENUMERATED
    {
    -- object bound troubles in a service
        other                               (0),
        object-undefined                    (1),  -- object not defined at the VDE
        object-class-inconsistent           (2),  -- class of object incompatible with asked service
        object-attribute-inconsistent       (3)   -- object attributes are inconsistent
    },
```

```
access                              [5] IMPLICIT ENUMERATED
{
-- object access error
    other                               (0),
    scope-of-access-violated            (1),  -- access denied through authorisation reason
    object-access-violated              (2),  -- access incompatible with object attribute
    hardware-fault                      (3),  -- access fail for hardware reason
    object-unavailable                  (4)   -- VDE hands object for unavailable
},

initiate                            [6] IMPLICIT ENUMERATED
{
-- initiate service error
    other                               (0),
    dlms-version-too-low                (1),  -- proposed DLMS version too low
    incompatible-conformance            (2),  -- proposed service not sufficient
    pdu-size-too-short                  (3),  -- proposed PDU size too short
    refused-by-the-VDE-Handler          (4)   -- vaa creation impossible or not allowed
},

load-data-set                       [7] IMPLICIT ENUMERATED
{
-- data set load services error
    other                               (0),
    primitive-out-of-sequence           (1),  -- according to the DataSet loading state transitions
    not-loadable                        (2),  -- loadable attribute set to FALSE
    dataset-size-too-large              (3),  -- evaluated Data Set size too large
    not-awaited-segment                 (4),  -- proposed segment not awaited
    interpretation-failure              (5),  -- segment interpretation error
    storage-failure                     (6),  -- segment storage error
    data-set-not-ready                  (7)   -- Data Set not in correct state for uploading
```

```
    },

    -- change-scope                   [8] IMPLICIT ENUMERATED

    task                              [9] IMPLICIT ENUMERATED
    {
    -- TI services error
        other                              (0),
        no-remote-control                  (1),  -- Remote Control parameter set to FALSE
        ti-stopped                         (2),  -- TI in stopped state
        ti-running                         (3),  -- TI in running state
        ti-unusable                        (4)   -- TI in unusable state
    }

    -- other                          [10] IMPLICIT ENUMERATED
}


-- COSEM APDUs using short name referencing

ReadRequest::= SEQUENCE OF Variable-Access-Specification

ReadResponse::= SEQUENCE OF CHOICE
{
    data                              [0] Data,
    data-access-error                 [1] IMPLICIT Data-Access-Result,
    data-block-result                 [2] IMPLICIT Data-Block-Result,
    block-number                      [3] IMPLICIT Unsigned16
}

WriteRequest ::= SEQUENCE
{
```

```
    variable-access-specification      SEQUENCE OF Variable-Access-Specification,
    list-of-data                       SEQUENCE OF Data
}

WriteResponse::= SEQUENCE OF CHOICE
{
    success                            [0] IMPLICIT NULL,
    data-access-error                  [1] IMPLICIT Data-Access-Result,
    block-number                       [2] Unsigned16
}

UnconfirmedWriteRequest::= SEQUENCE
{
    variable-access-specification      SEQUENCE OF Variable-Access-Specification,
    list-of-data                       SEQUENCE OF Data
}

InformationReportRequest::= SEQUENCE
{
    current-time                       GeneralizedTime OPTIONAL,
    variable-access-specification      SEQUENCE OF Variable-Access-Specification,
    list-of-data                       SEQUENCE OF Data
}


-- COSEM APDUs using logical name referencing

Get-Request::= CHOICE
{
    get-request-normal                 [1] IMPLICIT    Get-Request-Normal,
    get-request-next                   [2] IMPLICIT    Get-Request-Next,
    get-request-with-list              [3] IMPLICIT    Get-Request-With-List
```

```
}

Get-Request-Normal::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    cosem-attribute-descriptor         Cosem-Attribute-Descriptor,
    access-selection                   Selective-Access-Descriptor OPTIONAL
}

Get-Request-Next::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    block-number                       Unsigned32
}

Get-Request-With-List::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    attribute-descriptor-list          SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection
}

Get-Response::= CHOICE
{
    get-response-normal                [1] IMPLICIT    Get-Response-Normal,
    get-response-with-datablock        [2] IMPLICIT    Get-Response-With-Datablock,
    get-response-with-list             [3] IMPLICIT    Get-Response-With-List
}

Get-Response-Normal::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    result                             Get-Data-Result
```

```
}
Get-Response-With-Datablock::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    result                           DataBlock-G
}

Get-Response-With-List::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    result                           SEQUENCE OF Get-Data-Result
}

Set-Request::= CHOICE
{
  set-request-normal                         [1] IMPLICIT  Set-Request-Normal,
  set-request-with-first-datablock           [2] IMPLICIT  Set-Request-With-First-Datablock,
  set-request-with-datablock                 [3] IMPLICIT  Set-Request-With-Datablock,
  set-request-with-list                      [4] IMPLICIT  Set-Request-With-List,
  set-request-with-list-and-first-datablock  [5] IMPLICIT  Set-Request-With-List-And-First-Datablock
}

Set-Request-Normal::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    cosem-attribute-descriptor       Cosem-Attribute-Descriptor,
    access-selection                 Selective-Access-Descriptor OPTIONAL,
    value                            Data
}

Set-Request-With-First-Datablock::= SEQUENCE
{
```

```
    invoke-id-and-priority             Invoke-Id-And-Priority,
    cosem-attribute-descriptor         Cosem-Attribute-Descriptor,
    access-selection                   [0] IMPLICIT Selective-Access-Descriptor OPTIONAL,
    datablock                          DataBlock-SA
}

Set-Request-With-Datablock::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    datablock                          DataBlock-SA
}

Set-Request-With-List::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    attribute-descriptor-list          SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection,
    value-list                         SEQUENCE OF Data
}

Set-Request-With-List-And-First-Datablock::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    attribute-descriptor-list          SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection,
    datablock                          DataBlock-SA
}

Set-Response::= CHOICE
{
    set-response-normal                     [1] IMPLICIT   Set-Response-Normal,
    set-response-datablock                  [2] IMPLICIT   Set-Response-Datablock,
    set-response-last-datablock             [3] IMPLICIT   Set-Response-Last-Datablock,
    set-response-last-datablock-with-list   [4] IMPLICIT   Set-Response-Last-Datablock-With-List,
```

```
    set-response-with-list                  [5] IMPLICIT   Set-Response-With-List
}

Set-Response-Normal::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    result                             Data-Access-Result
}

Set-Response-Datablock::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    block-number                       Unsigned32
}

Set-Response-Last-Datablock::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    result                             Data-Access-Result,
    block-number                       Unsigned32
}

Set-Response-Last-Datablock-With-List::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    result                             SEQUENCE OF Data-Access-Result,
    block-number                       Unsigned32
}

Set-Response-With-List::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
```

```
    result                               SEQUENCE OF Data-Access-Result
}

Action-Request::= CHOICE
{
  action-request-normal                         [1] IMPLICIT Action-Request-Normal,
  action-request-next-pblock                    [2] IMPLICIT Action-Request-Next-Pblock,
  action-request-with-list                      [3] IMPLICIT Action-Request-With-List,
  action-request-with-first-pblock              [4] IMPLICIT Action-Request-With-First-Pblock,
  action-request-with-list-and-first-pblock     [5] IMPLICIT Action-Request-With-List-And-First-Pblock,
  action-request-with-pblock                    [6] IMPLICIT Action-Request-With-Pblock
}

Action-Request-Normal::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    cosem-method-descriptor             Cosem-Method-Descriptor,
    method-invocation-parameters        Data OPTIONAL
}

Action-Request-Next-Pblock::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    block-number                       Unsigned32
}

Action-Request-With-List::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    cosem-method-descriptor-list       SEQUENCE OF Cosem-Method-Descriptor,
    method-invocation-parameters       SEQUENCE OF Data
}
```

```
Action-Request-With-First-Pblock::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
    cosem-method-descriptor           Cosem-Method-Descriptor,
    pblock                            DataBlock-SA
}

Action-Request-With-List-And-First-Pblock::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
    cosem-method-descriptor-list      SEQUENCE OF Cosem-Method-Descriptor,
    pblock                            DataBlock-SA
}

Action-Request-With-Pblock::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
    pblock                            DataBlock-SA
}

Action-Response::= CHOICE
{
    action-response-normal            [1] IMPLICIT    Action-Response-Normal,
    action-response-with-pblock       [2] IMPLICIT    Action-Response-With-Pblock,
    action-response-with-list         [3] IMPLICIT    Action-Response-With-List,
    action-response-next-pblock       [4] IMPLICIT    Action-Response-Next-Pblock
}

Action-Response-Normal::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
```

```
    single-response                   Action-Response-With-Optional-Data
}

Action-Response-With-Pblock::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
    pblock                            DataBlock-SA
}

Action-Response-With-List::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
    list-of-responses                 SEQUENCE OF Action-Response-With-Optional-Data
}

Action-Response-Next-Pblock::= SEQUENCE
{
    invoke-id-and-priority            Invoke-Id-And-Priority,
    block-number                      Unsigned32
}

EventNotificationRequest::= SEQUENCE
{
    time                              OCTET STRING  OPTIONAL,
    cosem-attribute-descriptor        Cosem-Attribute-Descriptor,
    attribute-value                   Data
}

ExceptionResponse::= SEQUENCE
{
    state-error                       [0] IMPLICIT ENUMERATED
    {
```

```
        service-not-allowed                 (1),
        service-unknown                     (2)
    },
    service-error                       [1] IMPLICIT ENUMERATED
    {
        operation-not-possible              (1),
        service-not-supported               (2),
        other-reason                        (3)
    }
}


--      Access

Access-Request::= SEQUENCE
{
   long-invoke-id-and-priority         Long-Invoke-Id-And-Priority,
   date-time                           OCTET STRING,
   access-request-body                 Access-Request-Body
}

Access-Response::= SEQUENCE
{
       long-invoke-id-and-priority        Long-Invoke-Id-And-Priority,
       date-time                          OCTET STRING,
       access-response-body               Access-Response-Body
}


--      Data-Notification

Data-Notification::= SEQUENCE
```

```
{
   long-invoke-id-and-priority         Long-Invoke-Id-And-Priority,
   date-time                           OCTET STRING,
   notification-body                   Notification-Body
}
Data-Notification-Confirm ::= SEQUENCE
{
   long-invoke-id-and-priority         Long-Invoke-Id-And-Priority,
   date-time                           OCTET STRING
}
-- General APDUs

General-Ded-Ciphering::= SEQUENCE
{
   system-title                        OCTET STRING,
   ciphered-content                    OCTET STRING
}

General-Glo-Ciphering::= SEQUENCE
{
   system-title                        OCTET STRING,
   ciphered-content                    OCTET STRING
}

General-Ciphering::= SEQUENCE
{
   transaction-id                      OCTET STRING,
   originator-system-title             OCTET STRING,
   recipient-system-title              OCTET STRING,
   date-time                           OCTET STRING,
   other-information                   OCTET STRING,
   key-info                            Key-Info OPTIONAL,
```

```
   ciphered-content                    OCTET STRING
}

General-Signing::= SEQUENCE
{
   transaction-id                      OCTET STRING,
   originator-system-title             OCTET STRING,
   recipient-system-title              OCTET STRING,
   date-time                           OCTET STRING,
   other-information                   OCTET STRING,
   content                             OCTET STRING,
   signature                           OCTET STRING
}

General-Block-Transfer::= SEQUENCE
{
   block-control                       Block-Control,
   block-number                        Unsigned16,
   block-number-ack                    Unsigned16,
   block-data                          OCTET STRING
}


-- Types used in the xDLMS data transfer services

Variable-Access-Specification::= CHOICE
{
    variable-name                      [2] IMPLICIT ObjectName,
-- detailed-access [3] is not used in DLMS/COSEM
    parameterized-access               [4] IMPLICIT Parameterized-Access,
    block-number-access                [5] IMPLICIT Block-Number-Access,
    read-data-block-access             [6] IMPLICIT Read-Data-Block-Access,
```

```
    write-data-block-access          [7] IMPLICIT Write-Data-Block-Access
}

Parameterized-Access::= SEQUENCE
{
    variable-name                    ObjectName,
    selector                         Unsigned8,
    parameter                        Data
}

Block-Number-Access::= SEQUENCE
{
    block-number                     Unsigned16
}

Read-Data-Block-Access::= SEQUENCE
{
    last-block                       BOOLEAN,
    block-number                     Unsigned16,
    raw-data                         OCTET STRING
}


Write-Data-Block-Access::= SEQUENCE
{
   last-block                        BOOLEAN,
   block-number                      Unsigned16
}

Data::= CHOICE
{
    null-data                        [0]   IMPLICIT   NULL,
```

```
array                          [1]   IMPLICIT   SEQUENCE OF Data,
structure                      [2]   IMPLICIT   SEQUENCE OF Data,
boolean                        [3]   IMPLICIT   BOOLEAN,
bit-string                     [4]   IMPLICIT   BIT STRING,
double-long                    [5]   IMPLICIT   Integer32,
double-long-unsigned           [6]   IMPLICIT   Unsigned32,
octet-string                   [9]   IMPLICIT   OCTET STRING,
visible-string                 [10]  IMPLICIT   VisibleString,
utf8-string                    [12]  IMPLICIT   UTF8String,
bcd                            [13]  IMPLICIT   Integer8,
integer                        [15]  IMPLICIT   Integer8,
long                           [16]  IMPLICIT   Integer16,
unsigned                       [17]  IMPLICIT   Unsigned8,
long-unsigned                  [18]  IMPLICIT   Unsigned16,
compact-array                  [19]  IMPLICIT   SEQUENCE
{
    contents-description           [0]              TypeDescription,
    array-contents                 [1]   IMPLICIT   OCTET STRING
},
long64                         [20]  IMPLICIT   Integer64,
long64-unsigned                [21]  IMPLICIT   Unsigned64,
enum                           [22]  IMPLICIT   Unsigned8,
float32                        [23]  IMPLICIT   OCTET STRING (SIZE(4)),
float64                        [24]  IMPLICIT   OCTET STRING (SIZE(8)),
date-time                      [25]  IMPLICIT   OCTET STRING (SIZE(12)),
date                           [26]  IMPLICIT   OCTET STRING (SIZE(5)),
time                           [27]  IMPLICIT   OCTET STRING (SIZE(4)),
delta-integer                  [28]  IMPLICIT   Integer8,
delta-long                     [29]  IMPLICIT   Integer16,
delta-double-long              [30]  IMPLICIT   Integer32,
delta-unsigned                 [31]  IMPLICIT   Unsigned8,
delta-long-unsigned            [32]  IMPLICIT   Unsigned16,
delta-double-long-unsigned     [33]  IMPLICIT   Unsigned32,

dont-care                      [255] IMPLICIT   NULL
```

```
}

-- The following TypeDescription relates to the compact-array data Type

TypeDescription::= CHOICE
{
    null-data                          [0]   IMPLICIT  NULL,
    array                              [1]   IMPLICIT  SEQUENCE
    {
        number-of-elements        Unsigned16,
        type-description          TypeDescription
    },
    structure                          [2]   IMPLICIT  SEQUENCE OF TypeDescription,
    boolean                            [3]   IMPLICIT  NULL,
    bit-string                         [4]   IMPLICIT  NULL,
    double-long                        [5]   IMPLICIT  NULL,
    double-long-unsigned               [6]   IMPLICIT  NULL,
    octet-string                       [9]   IMPLICIT  NULL,
    visible-string                     [10]  IMPLICIT  NULL,
    utf8-string                        [12]  IMPLICIT  NULL,
    bcd                                [13]  IMPLICIT  NULL,
    integer                            [15]  IMPLICIT  NULL,
    long                               [16]  IMPLICIT  NULL,
    unsigned                           [17]  IMPLICIT  NULL,
    long-unsigned                      [18]  IMPLICIT  NULL,
    long64                             [20]  IMPLICIT  NULL,
    long64-unsigned                    [21]  IMPLICIT  NULL,
    enum                               [22]  IMPLICIT  NULL,
    float32                            [23]  IMPLICIT  NULL,
    float64                            [24]  IMPLICIT  NULL,
    date-time                          [25]  IMPLICIT  NULL,
    date                               [26]  IMPLICIT  NULL,
```

```
    time                              [27]  IMPLICIT  NULL,
    dont-care                         [255] IMPLICIT  NULL
}

Data-Access-Result::= ENUMERATED
{
    success                           (0),
    hardware-fault                    (1),
    temporary-failure                 (2),
    read-write-denied                 (3),
    object-undefined                  (4),
    object-class-inconsistent         (9),
    object-unavailable                (11),
    type-unmatched                    (12),
    scope-of-access-violated          (13),
    data-block-unavailable            (14),
    long-get-aborted                  (15),
    no-long-get-in-progress           (16),
    long-set-aborted                  (17),
    no-long-set-in-progress           (18),
    data-block-number-invalid         (19),
    other-reason                      (250)
}

Action-Result::= ENUMERATED
{
    success                           (0),
    hardware-fault                    (1),
    temporary-failure                 (2),
    read-write-denied                 (3),
    object-undefined                  (4),
    object-class-inconsistent         (9),
```

```
    object-unavailable                 (11),
    type-unmatched                     (12),
    scope-of-access-violated           (13),
    data-block-unavailable             (14),
    long-action-aborted                (15),
    no-long-action-in-progress         (16),
    other-reason                       (250)
}

-- IEC 61334-6:2000 Clause 5 specifies that bits of any byte are numbered from 1 to 8,
-- where bit 8 is the most significant.
-- In IEC 62056-5-3, bits are numbered from 0 to 7.
-- Use of Invoke-Id-And-Priority
--    invoke-id                bits 0-3
--    reserved                 bits 4-5
--    service-class            bit  6        0 = Unconfirmed, 1 = Confirmed
--    priority                 bit  7        0 = Normal, 1 = High
Invoke-Id-And-Priority::=            Unsigned8

-- Use of Long-Invoke-Id-And-Priority
--    long-invoke-id           bits 0-23
--    reserved                 bits 24-27
--    self-descriptive         bit  28       0 = Not-Self-Descriptive, 1 = Self-Descriptive
--    processing-option        bit  29       0 = Continue on Error, 1 = Break on Error
--    service-class            bit  30       0 = Unconfirmed, 1 = Confirmed
--    priority                 bit  31       0 = Normal, 1 = High
Long-Invoke-Id-And-Priority::=       Unsigned32

Cosem-Attribute-Descriptor::= SEQUENCE
{
    class-id                           Cosem-Class-Id,
    instance-id                        Cosem-Object-Instance-Id,
```

```
    attribute-id                       Cosem-Object-Attribute-Id
}

Cosem-Method-Descriptor::= SEQUENCE
{
    class-id                           Cosem-Class-Id,
    instance-id                        Cosem-Object-Instance-Id,
    method-id                          Cosem-Object-Method-Id
}

Cosem-Class-Id::=                      Unsigned16

Cosem-Object-Instance-Id::=            OCTET STRING (SIZE(6))

Cosem-Object-Attribute-Id::=           Integer8

Cosem-Object-Method-Id::=              Integer8

Selective-Access-Descriptor::= SEQUENCE
{
    access-selector                    Unsigned8,
    access-parameters                  Data
}

Cosem-Attribute-Descriptor-With-Selection::= SEQUENCE
{
    cosem-attribute-descriptor         Cosem-Attribute-Descriptor,
    access-selection                   Selective-Access-Descriptor OPTIONAL
}

Get-Data-Result::= CHOICE
{
```

```
    data                                    [0] Data,
    data-access-result                      [1] IMPLICIT Data-Access-Result
}

Data-Block-Result::= SEQUENCE  -- Used in ReadResponse with block transfer
{
    last-block                             BOOLEAN,
    block-number                           Unsigned16,
    raw-data                               OCTET STRING
}

DataBlock-G::= SEQUENCE     -- G == DataBlock for the GET-response
{
    last-block                             BOOLEAN,
    block-number                           Unsigned32,
    result  CHOICE
    {
        raw-data                           [0] IMPLICIT OCTET STRING,
        data-access-result                 [1] IMPLICIT Data-Access-Result
    }
}

DataBlock-SA::= SEQUENCE    -- SA == DataBlock for the SET-request, ACTION-request and ACTION-response
{
    last-block                             BOOLEAN,
    block-number                           Unsigned32,
    raw-data                               OCTET STRING
}

Action-Response-With-Optional-Data::= SEQUENCE
{
    result                                 Action-Result,
```

```
    return-parameters                Get-Data-Result  OPTIONAL
}

Notification-Body::= SEQUENCE
{
   data-value                         Data
}

List-Of-Data::= SEQUENCE OF Data

Access-Request-Get::= SEQUENCE
{
   cosem-attribute-descriptor         Cosem-Attribute-Descriptor
}

Access-Request-Get-With-Selection::= SEQUENCE
{
   cosem-attribute-descriptor         Cosem-Attribute-Descriptor,
   access-selection                   Selective-Access-Descriptor
}

Access-Request-Set::= SEQUENCE
{
   cosem-attribute-descriptor         Cosem-Attribute-Descriptor
}

Access-Request-Set-With-Selection::= SEQUENCE
{
   cosem-attribute-descriptor         Cosem-Attribute-Descriptor,
   access-selection                   Selective-Access-Descriptor
}
```

```
Access-Request-Action::= SEQUENCE
{
   cosem-method-descriptor             Cosem-Method-Descriptor
}

Access-Request-Specification::= CHOICE
{
   access-request-get                  [1] Access-Request-Get,
   access-request-set                  [2] Access-Request-Set,
   access-request-action               [3] Access-Request-Action,
   access-request-get-with-selection   [4] Access-Request-Get-With-Selection,
   access-request-set-with-selection   [5] Access-Request-Set-With-Selection
}

List-Of-Access-Request-Specification::= SEQUENCE OF Access-Request-Specification

Access-Request-Body::= SEQUENCE
{
   access-request-specification        List-Of-Access-Request-Specification,
   access-request-list-of-data         List-Of-Data
}

Access-Response-Get::= SEQUENCE
{
   result                              Data-Access-Result
}

Access-Response-Set::= SEQUENCE
{
   result                              Data-Access-Result
}
```

```
Access-Response-Action::= SEQUENCE
{
   result                                Action-Result
}

Access-Response-Specification::= CHOICE
{
   access-response-get                   [1] Access-Response-Get,
   access-response-set                   [2] Access-Response-Set,
   access-response-action                [3] Access-Response-Action
}

List-Of-Access-Response-Specification::= SEQUENCE OF Access-Response-Specification

Access-Response-Body::= SEQUENCE
{
   access-request-specification          [0] List-Of-Access-Request-Specification OPTIONAL,
   access-response-list-of-data          List-Of-Data,
   access-response-specification         List-Of-Access-Response-Specification
}

-- Key-info

Key-Id::= ENUMERATED
{
       global-unicast-encryption-key      (0),
       global-broadcast-encryption-key    (1)
}

Kek-Id::= ENUMERATED
{
       master-key                         (0)
```

```
}


Identified-Key::= SEQUENCE
{
       key-id                              Key-Id
}

Wrapped-Key::= SEQUENCE
{
    kek-id                              Kek-Id,
    key-ciphered-data                   OCTET STRING
}

Agreed-Key::= SEQUENCE
{
    key-parameters                      OCTET STRING,
    key-ciphered-data                   OCTET STRING
}

Key-Info::= CHOICE
{
    identified-key                      [0] Identified-Key,
    wrapped-key                         [1] Wrapped-Key,
    agreed-key                          [2] Agreed-Key,
}

-- Use of Block-Control
--    window                  bits 0-5     window advertise
--    streaming               bit  6       0 = No Streaming active, 1 = Streaming active
--    last-block              bit  7       0 = Not Last Block, 1 = Last Block
Block-Control::=                       Unsigned8
```

```
END
```

9 COSEM APDU XML schema

9.1 General

ITU-T recommendations X.693 and X.694 provide XML encoding rules to Abstract Syntax Notation 1 (ASN.1) and XML Schema Definitions Language (XSD) mapping to ASN.1. No recommendation is provided to map ASN.1 to XSD. In this Clause 9 COSEMpdu ASN.1 definition is provided and mapped to COSEMpdu XSD definition.

XML has gained wide acceptance in the IT industry. The purpose of such encoding is to enable transfer of COSEM model content with various means in the form of XML encoded content. It can be a XML document exchanged between applications, content included in Web services SOAP messages or content encapsulated in e-mail messages, to name just few of the applications. Interoperability and mapping between ASN.1 encoded APDUs and XML encoded content is important in both directions. On one side ASN.1 has enabled creation of XML encoded content with support for XML Encoding Rules (See ITU-T X.693 and ITU-T X.694). On the other hand IT industry is searching for solutions for optimal transfer of XML content with XML optimized packaging. For that purposes conversion between W3C XML Schema and ASN.1 definition is crucial. Conversion in both directions enables proper conversion of XML content to ASN.1 encoded content and vice versa. Subclause 9.2 contains mapping of COSEMPdu ASN.1 definition into COSEMPdu XML Schema (XSD).

9.2 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.dlms.com/COSEMpdu"
            targetNamespace="http://www.dlms.com/COSEMpdu"
            elementFormDefault="qualified">

   <!-- ASN.1 definitions -->
   <xsd:complexType name="NULL" final="#all" />

   <xsd:simpleType name="BitString">
       <xsd:restriction base="xsd:string">
           <xsd:pattern value="[0-1]{0,}" />
       </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="ObjectIdentifier">
```

```
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="[0-2](\.[1-3]?[0-9]?(\.\d+)*)?" />
    </xsd:restriction>
</xsd:simpleType>


<!-- ACSE-APDU definition -->
<xsd:element name="aCSE-APDU" type="ACSE-APDU"/>
<xsd:complexType name="ACSE-APDU">
   <xsd:choice>
      <xsd:element name="aarq" type="AARQ-apdu"/>
      <xsd:element name="aare" type="AARE-apdu"/>
      <xsd:element name="rlrq" type="RLRQ-apdu"/>
      <xsd:element name="rlre" type="RLRE-apdu"/>
   </xsd:choice>
</xsd:complexType>

<!-- xDLMS-APDU definition -->
<xsd:element name="xDLMS-APDU" type="XDLMS-APDU"/>
<xsd:complexType name="XDLMS-APDU">
   <xsd:choice>
      <xsd:element name="initiateRequest" type="InitiateRequest"/>
      <xsd:element name="readRequest" type="ReadRequest"/>
      <xsd:element name="writeRequest" type="WriteRequest"/>
      <xsd:element name="initiateResponse" type="InitiateResponse"/>
      <xsd:element name="readResponse" type="ReadResponse"/>
      <xsd:element name="writeResponse" type="WriteResponse"/>
      <xsd:element name="confirmedServiceError" type="ConfirmedServiceError"/>
      <xsd:element name="data-notification" type="Data-Notification"/>
      <xsd:element name="data-notification-confirm" type="Data-Notification-Confirm"/>
      <xsd:element name="unconfirmedWriteRequest" type="UnconfirmedWriteRequest"/>
      <xsd:element name="informationReportRequest" type="InformationReportRequest"/>
```

```
<xsd:element name="glo-initiateRequest" type="xsd:hexBinary"/>
<xsd:element name="glo-readRequest" type="xsd:hexBinary"/>
<xsd:element name="glo-writeRequest" type="xsd:hexBinary"/>
<xsd:element name="glo-initiateResponse" type="xsd:hexBinary"/>
<xsd:element name="glo-readResponse" type="xsd:hexBinary"/>
<xsd:element name="glo-writeResponse" type="xsd:hexBinary"/>
<xsd:element name="glo-confirmedServiceError" type="xsd:hexBinary"/>
<xsd:element name="glo-unconfirmedWriteRequest" type="xsd:hexBinary"/>
<xsd:element name="glo-informationReportRequest" type="xsd:hexBinary"/>
<xsd:element name="ded-initiateRequest" type="xsd:hexBinary"/>
<xsd:element name="ded-readRequest" type="xsd:hexBinary"/>
<xsd:element name="ded-writeRequest" type="xsd:hexBinary"/>
<xsd:element name="ded-initiateResponse" type="xsd:hexBinary"/>
<xsd:element name="ded-readResponse" type="xsd:hexBinary"/>
<xsd:element name="ded-writeResponse" type="xsd:hexBinary"/>
<xsd:element name="ded-confirmedServiceError" type="xsd:hexBinary"/>
<xsd:element name="ded-unconfirmedWriteRequest" type="xsd:hexBinary"/>
<xsd:element name="ded-informationReportRequest" type="xsd:hexBinary"/>
<xsd:element name="get-request" type="Get-Request"/>
<xsd:element name="set-request" type="Set-Request"/>
<xsd:element name="event-notification-request" type="EventNotificationRequest"/>
<xsd:element name="action-request" type="Action-Request"/>
<xsd:element name="get-response" type="Get-Response"/>
<xsd:element name="set-response" type="Set-Response"/>
<xsd:element name="action-response" type="Action-Response"/>
<xsd:element name="glo-get-request" type="xsd:hexBinary"/>
<xsd:element name="glo-set-request" type="xsd:hexBinary"/>
<xsd:element name="glo-event-notification-request" type="xsd:hexBinary"/>
<xsd:element name="glo-action-request" type="xsd:hexBinary"/>
<xsd:element name="glo-get-response" type="xsd:hexBinary"/>
<xsd:element name="glo-set-response" type="xsd:hexBinary"/>
<xsd:element name="glo-action-response" type="xsd:hexBinary"/>
```

```
            <xsd:element name="ded-get-request" type="xsd:hexBinary"/>
            <xsd:element name="ded-set-request" type="xsd:hexBinary"/>
            <xsd:element name="ded-event-notification-request" type="xsd:hexBinary"/>
            <xsd:element name="ded-actionRequest" type="xsd:hexBinary"/>
            <xsd:element name="ded-get-response" type="xsd:hexBinary"/>
            <xsd:element name="ded-set-response" type="xsd:hexBinary"/>
            <xsd:element name="ded-action-response" type="xsd:hexBinary"/>
            <xsd:element name="exception-response" type="ExceptionResponse"/>
            <xsd:element name="access-request" type="Access-Request"/>
            <xsd:element name="access-response" type="Access-Response"/>
            <xsd:element name="general-glo-ciphering" type="General-Glo-Ciphering"/>
            <xsd:element name="general-ded-ciphering" type="General-Ded-Ciphering"/>
            <xsd:element name="general-ciphering" type="General-Ciphering"/>
            <xsd:element name="general-signing" type="General-Signing"/>
            <xsd:element name="general-block-transfer" type="General-Block-Transfer"/>
        </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="Data-Notification-Confirm">
        <xsd:sequence>
            <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
            <xsd:element name="date-time" type="xsd:hexBinary"/>
        </xsd:sequence>
     </xsd:complexType>

    <xsd:simpleType name="Application-context-name">
        <xsd:restriction base="ObjectIdentifier"/>
    </xsd:simpleType>

    <xsd:simpleType name="AP-title">
        <xsd:restriction base="xsd:hexBinary"/>
    </xsd:simpleType>

```

```xml
<xsd:simpleType name="AE-qualifier">
   <xsd:restriction base="xsd:hexBinary"/>
</xsd:simpleType>

<xsd:simpleType name="AP-invocation-identifier">
   <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="AE-invocation-identifier">
   <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="ACSE-requirements">
   <xsd:union memberTypes="BitString">
   <xsd:simpleType>
      <xsd:list>
         <xsd:simpleType>
            <xsd:restriction base="xsd:token">
               <xsd:enumeration value="authentication"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:list>
   </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Mechanism-name">
   <xsd:restriction base="ObjectIdentifier"/>
</xsd:simpleType>

<xsd:simpleType name="Implementation-data">
   <xsd:restriction base="xsd:string"/>
```

```
    </xsd:simpleType>

    <xsd:simpleType name="Association-information">
        <xsd:restriction base="xsd:hexBinary"/>
    </xsd:simpleType>

    <xsd:simpleType name="Association-result">
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="xsd:token">
                    <xsd:enumeration value="accepted"/>
                    <xsd:enumeration value="rejected-permanent"/>
                    <xsd:enumeration value="rejected-transient"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="xsd:integer"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>

    <xsd:simpleType name="Release-request-reason">
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="xsd:token">
                    <xsd:enumeration value="normal"/>
                    <xsd:enumeration value="urgent"/>
                    <xsd:enumeration value="user-defined"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="xsd:integer"/>
```

```
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Release-response-reason">
   <xsd:union>
      <xsd:simpleType>
         <xsd:restriction base="xsd:token">
            <xsd:enumeration value="normal"/>
            <xsd:enumeration value="not-finished"/>
            <xsd:enumeration value="user-defined"/>
         </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
         <xsd:restriction base="xsd:integer"/>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Integer8">
   <xsd:restriction base="xsd:byte"/>
</xsd:simpleType>

<xsd:simpleType name="Integer16">
   <xsd:restriction base="xsd:short"/>
</xsd:simpleType>

<xsd:simpleType name="Integer32">
   <xsd:restriction base="xsd:int"/>
</xsd:simpleType>

<xsd:simpleType name="Integer64">
```

```
        <xsd:restriction base="xsd:long"/>
    </xsd:simpleType>

    <xsd:simpleType name="Unsigned8">
        <xsd:restriction base="xsd:unsignedByte"/>
    </xsd:simpleType>

    <xsd:simpleType name="Unsigned16">
        <xsd:restriction base="xsd:unsignedShort"/>
    </xsd:simpleType>

    <xsd:simpleType name="Unsigned32">
        <xsd:restriction base="xsd:unsignedInt"/>
    </xsd:simpleType>

    <xsd:simpleType name="Unsigned64">
        <xsd:restriction base="xsd:unsignedLong"/>
    </xsd:simpleType>

    <xsd:simpleType name="Conformance">
        <xsd:union memberTypes="BitString">
        <xsd:simpleType>
            <xsd:list>
                <xsd:simpleType>
                    <xsd:restriction base="xsd:token">
                        <xsd:enumeration value="reserved-zero"/>
                        <xsd:enumeration value="general-protection"/>
                        <xsd:enumeration value="general-block-transfer"/>
                        <xsd:enumeration value="read"/>
                        <xsd:enumeration value="write"/>
                        <xsd:enumeration value="unconfirmed-write"/>
                        <xsd:enumeration value="delta-value-encoding"/>
```

```
               <xsd:enumeration value="reserved-seven"/>
               <xsd:enumeration value="attribute0-supported-with-set"/>
               <xsd:enumeration value="priority-mgmt-supported"/>
               <xsd:enumeration value="attribute0-supported-with-get"/>
               <xsd:enumeration value="block-transfer-with-get-or-read"/>
               <xsd:enumeration value="block-transfer-with-set-or-write"/>
               <xsd:enumeration value="block-transfer-with-action"/>
               <xsd:enumeration value="multiple-references"/>
               <xsd:enumeration value="information-report"/>
               <xsd:enumeration value="data-notification"/>
               <xsd:enumeration value="access"/>
               <xsd:enumeration value="parameterized-access"/>
               <xsd:enumeration value="get"/>
               <xsd:enumeration value="set"/>
               <xsd:enumeration value="selective-access"/>
               <xsd:enumeration value="event-notification"/>
               <xsd:enumeration value="action"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:list>
   </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="ObjectName">
   <xsd:restriction base="Integer16"/>
</xsd:simpleType>

<xsd:simpleType name="Data-Access-Result">
   <xsd:restriction base="xsd:token">
      <xsd:enumeration value="success"/>
      <xsd:enumeration value="hardware-fault"/>
```

```
            <xsd:enumeration value="temporary-failure"/>
            <xsd:enumeration value="read-write-denied"/>
            <xsd:enumeration value="object-undefined"/>
            <xsd:enumeration value="object-class-inconsistent"/>
            <xsd:enumeration value="object-unavailable"/>
            <xsd:enumeration value="type-unmatched"/>
            <xsd:enumeration value="scope-of-access-violated"/>
            <xsd:enumeration value="data-block-unavailable"/>
            <xsd:enumeration value="long-get-aborted"/>
            <xsd:enumeration value="no-long-get-in-progress"/>
            <xsd:enumeration value="long-set-aborted"/>
            <xsd:enumeration value="no-long-set-in-progress"/>
            <xsd:enumeration value="data-block-number-invalid"/>
            <xsd:enumeration value="other-reason"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Action-Result">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="success"/>
            <xsd:enumeration value="hardware-fault"/>
            <xsd:enumeration value="temporary-failure"/>
            <xsd:enumeration value="read-write-denied"/>
            <xsd:enumeration value="object-undefined"/>
            <xsd:enumeration value="object-class-inconsistent"/>
            <xsd:enumeration value="object-unavailable"/>
            <xsd:enumeration value="type-unmatched"/>
            <xsd:enumeration value="scope-of-access-violated"/>
            <xsd:enumeration value="data-block-unavailable"/>
            <xsd:enumeration value="long-action-aborted"/>
            <xsd:enumeration value="no-long-action-in-progress"/>
            <xsd:enumeration value="other-reason"/>
```

```
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="Invoke-Id-And-Priority">
      <xsd:restriction base="Unsigned8"/>
   </xsd:simpleType>

   <xsd:simpleType name="Long-Invoke-Id-And-Priority">
      <xsd:restriction base="Unsigned32"/>
   </xsd:simpleType>

   <xsd:simpleType name="Cosem-Class-Id">
      <xsd:restriction base="Unsigned16"/>
   </xsd:simpleType>

   <xsd:simpleType name="Cosem-Object-Instance-Id">
      <xsd:restriction base="xsd:hexBinary">
         <xsd:length value="6"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="Cosem-Object-Attribute-Id">
      <xsd:restriction base="Integer8"/>
   </xsd:simpleType>

   <xsd:simpleType name="Cosem-Object-Method-Id">
      <xsd:restriction base="Integer8"/>
   </xsd:simpleType>

   <xsd:simpleType name="Key-Id">
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="global-unicast-encryption-key"/>
```

```
            <xsd:enumeration value="global-broadcast-encryption-key"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Kek-Id">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="master-key"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Block-Control">
        <xsd:restriction base="Unsigned8"/>
    </xsd:simpleType>

    <xsd:complexType name="Authentication-value">
        <xsd:choice>
            <xsd:element name="charstring" type="xsd:string"/>
            <xsd:element name="bitstring" type="BitString"/>
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="AARQ-apdu">
        <xsd:sequence>
            <xsd:element name="protocol-version" minOccurs="0">
                <xsd:simpleType>
                    <xsd:union memberTypes="BitString">
                    <xsd:simpleType>
                        <xsd:list>
                            <xsd:simpleType>
                                <xsd:restriction base="xsd:token">
                                    <xsd:enumeration value="version1"/>
                                </xsd:restriction>
```

```
                  </xsd:simpleType>
               </xsd:list>
            </xsd:simpleType>
            </xsd:union>
         </xsd:simpleType>
      </xsd:element>
      <xsd:element name="application-context-name" type="Application-context-name"/>
      <xsd:element name="called-AP-title" minOccurs="0" type="AP-title"/>
      <xsd:element name="called-AE-qualifier" minOccurs="0" type="AE-qualifier"/>
      <xsd:element name="called-AP-invocation-id" minOccurs="0" type="AP-invocation-identifier"/>
      <xsd:element name="called-AE-invocation-id" minOccurs="0" type="AE-invocation-identifier"/>
      <xsd:element name="calling-AP-title" minOccurs="0" type="AP-title"/>
      <xsd:element name="calling-AE-qualifier" minOccurs="0" type="AE-qualifier"/>
      <xsd:element name="calling-AP-invocation-id" minOccurs="0" type="AP-invocation-identifier"/>
      <xsd:element name="calling-AE-invocation-id" minOccurs="0" type="AE-invocation-identifier"/>
      <xsd:element name="sender-acse-requirements" minOccurs="0" type="ACSE-requirements"/>
      <xsd:element name="mechanism-name" minOccurs="0" type="Mechanism-name"/>
      <xsd:element name="calling-authentication-value" minOccurs="0" type="Authentication-value"/>
      <xsd:element name="implementation-information" minOccurs="0" type="Implementation-data"/>
      <xsd:element name="user-information" minOccurs="0" type="Association-information"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Associate-source-diagnostic">
   <xsd:choice>
      <xsd:element name="acse-service-user">
         <xsd:simpleType>
            <xsd:union>
               <xsd:simpleType>
                  <xsd:restriction base="xsd:token">
                     <xsd:enumeration value="null"/>
                     <xsd:enumeration value="no-reason-given"/>
```

```
                    <xsd:enumeration value="application-context-name-not-supported"/>
                    <xsd:enumeration value="calling-AP-title-not-recognized"/>
                    <xsd:enumeration value="calling-AP-invocation-identifier-not-recognized"/>
                    <xsd:enumeration value="calling-AE-qualifier-not-recognized"/>
                    <xsd:enumeration value="calling-AE-invocation-identifier-not-recognized"/>
                    <xsd:enumeration value="called-AP-title-not-recognized"/>
                    <xsd:enumeration value="called-AP-invocation-identifier-not-recognized"/>
                    <xsd:enumeration value="called-AE-qualifier-not-recognized"/>
                    <xsd:enumeration value="called-AE-invocation-identifier-not-recognized"/>
                    <xsd:enumeration value="authentication-mechanism-name-not-recognised"/>
                    <xsd:enumeration value="authentication-mechanism-name-required"/>
                    <xsd:enumeration value="authentication-failure"/>
                    <xsd:enumeration value="authentication-required"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="xsd:integer"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="acse-service-provider">
    <xsd:simpleType>
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="xsd:token">
                    <xsd:enumeration value="null"/>
                    <xsd:enumeration value="no-reason-given"/>
                    <xsd:enumeration value="no-common-acse-version"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
```

```
                            <xsd:restriction base="xsd:integer"/>
                        </xsd:simpleType>
                    </xsd:union>
                </xsd:simpleType>
            </xsd:element>
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="AARE-apdu">
        <xsd:sequence>
            <xsd:element name="protocol-version" minOccurs="0">
                <xsd:simpleType>
                    <xsd:union memberTypes="BitString">
                    <xsd:simpleType>
                        <xsd:list>
                            <xsd:simpleType>
                                <xsd:restriction base="xsd:token">
                                    <xsd:enumeration value="version1"/>
                                </xsd:restriction>
                            </xsd:simpleType>
                        </xsd:list>
                    </xsd:simpleType>
                    </xsd:union>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="application-context-name" type="Application-context-name"/>
            <xsd:element name="result" type="Association-result"/>
            <xsd:element name="result-source-diagnostic" type="Associate-source-diagnostic"/>
            <xsd:element name="responding-AP-title" minOccurs="0" type="AP-title"/>
            <xsd:element name="responding-AE-qualifier" minOccurs="0" type="AE-qualifier"/>
            <xsd:element name="responding-AP-invocation-id" minOccurs="0" type="AP-invocation-
identifier"/>
```

```xml
            <xsd:element name="responding-AE-invocation-id" minOccurs="0" type="AE-invocation-identifier"/>
            <xsd:element name="responder-acse-requirements" minOccurs="0" type="ACSE-requirements"/>
            <xsd:element name="mechanism-name" minOccurs="0" type="Mechanism-name"/>
            <xsd:element name="responding-authentication-value" minOccurs="0" type="Authentication-value"/>
            <xsd:element name="implementation-information" minOccurs="0" type="Implementation-data"/>
            <xsd:element name="user-information" minOccurs="0" type="Association-information"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="RLRQ-apdu">
         <xsd:sequence>
            <xsd:element name="reason" minOccurs="0" type="Release-request-reason"/>
            <xsd:element name="user-information" minOccurs="0" type="Association-information"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="RLRE-apdu">
         <xsd:sequence>
            <xsd:element name="reason" minOccurs="0" type="Release-response-reason"/>
            <xsd:element name="user-information" minOccurs="0" type="Association-information"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="InitiateRequest">
         <xsd:sequence>
            <xsd:element name="dedicated-key" minOccurs="0" type="xsd:hexBinary"/>
            <xsd:element name="response-allowed" default="true" type="xsd:boolean"/>
            <xsd:element name="proposed-quality-of-service" minOccurs="0" type="Integer8"/>
            <xsd:element name="proposed-dlms-version-number" type="Unsigned8"/>
            <xsd:element name="proposed-conformance" type="Conformance"/>
            <xsd:element name="client-max-receive-pdu-size" type="Unsigned16"/>
```

```
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TypeDescription">
    <xsd:choice>
      <xsd:element name="null-data" type="NULL"/>
      <xsd:element name="array">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="number-of-elements" type="Unsigned16"/>
            <xsd:element name="type-description" type="TypeDescription"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="structure">
        <xsd:complexType>
          <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="TypeDescription" type="TypeDescription"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="boolean" type="NULL"/>
      <xsd:element name="bit-string" type="NULL"/>
      <xsd:element name="double-long" type="NULL"/>
      <xsd:element name="double-long-unsigned" type="NULL"/>
      <xsd:element name="octet-string" type="NULL"/>
      <xsd:element name="visible-string" type="NULL"/>
      <xsd:element name="utf8-string" type="NULL"/>
      <xsd:element name="bcd" type="NULL"/>
      <xsd:element name="integer" type="NULL"/>
      <xsd:element name="long" type="NULL"/>
      <xsd:element name="unsigned" type="NULL"/>
```

```
            <xsd:element name="long-unsigned" type="NULL"/>
            <xsd:element name="long64" type="NULL"/>
            <xsd:element name="long64-unsigned" type="NULL"/>
            <xsd:element name="enum" type="NULL"/>
            <xsd:element name="float32" type="NULL"/>
            <xsd:element name="float64" type="NULL"/>
            <xsd:element name="date-time" type="NULL"/>
            <xsd:element name="date" type="NULL"/>
            <xsd:element name="time" type="NULL"/>
            <xsd:element name="dont-care" type="NULL"/>
         </xsd:choice>
      </xsd:complexType>

       <xsd:complexType name="SequenceOfData">
           <xsd:choice minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="null-data" type="NULL"/>
               <xsd:element name="array" type="SequenceOfData"/>
               <xsd:element name="structure" type="SequenceOfData"/>
               <xsd:element name="boolean" type="xsd:boolean"/>
               <xsd:element name="bit-string" type="BitString"/>
               <xsd:element name="double-long" type="Integer32"/>
               <xsd:element name="double-long-unsigned" type="Unsigned32"/>
               <xsd:element name="octet-string" type="xsd:hexBinary"/>
               <xsd:element name="visible-string" type="xsd:string"/>
               <xsd:element name="utf8-string" type="xsd:string"/>
               <xsd:element name="bcd" type="Integer8"/>
               <xsd:element name="integer" type="Integer8"/>
               <xsd:element name="long" type="Integer16"/>
               <xsd:element name="unsigned" type="Unsigned8"/>
               <xsd:element name="long-unsigned" type="Unsigned16"/>
               <xsd:element name="compact-array">
                   <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element name="contents-description" type="TypeDescription"/>
                    <xsd:element name="array-contents" type="xsd:hexBinary"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="long64" type="Integer64"/>
        <xsd:element name="long64-unsigned" type="Unsigned64"/>
        <xsd:element name="enum" type="Unsigned8"/>
        <xsd:element name="float32" type="xsd:float"/>
        <xsd:element name="float64" type="xsd:double"/>
        <xsd:element name="date-time">
            <xsd:simpleType>
                <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="12"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="date">
            <xsd:simpleType>
                <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="5"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="time">
            <xsd:simpleType>
                <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="4"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
```

```
            <xsd:element name="delta-integer" type="Integer8"/>
            <xsd:element name="delta-long" type="Integer16"/>
            <xsd:element name="delta-double-long" type="Integer32"/>
            <xsd:element name="delta-unsigned" type="Unsigned8"/>
            <xsd:element name="delta-long-unsigned" type="Unsigned16"/>
            <xsd:element name="delta-double-long-unsigned" type="Unsigned32"/>
            <xsd:element name="dont-care" type="NULL"/>
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="Data">
      <xsd:choice>
         <xsd:element name="null-data" type="NULL"/>
         <xsd:element name="array" type="SequenceOfData"/>
         <xsd:element name="structure" type="SequenceOfData"/>
         <xsd:element name="boolean" type="xsd:boolean"/>
         <xsd:element name="bit-string" type="BitString"/>
         <xsd:element name="double-long" type="Integer32"/>
         <xsd:element name="double-long-unsigned" type="Unsigned32"/>
         <xsd:element name="octet-string" type="xsd:hexBinary"/>
         <xsd:element name="visible-string" type="xsd:string"/>
         <xsd:element name="utf8-string" type="xsd:string"/>
         <xsd:element name="bcd" type="Integer8"/>
         <xsd:element name="integer" type="Integer8"/>
         <xsd:element name="long" type="Integer16"/>
         <xsd:element name="unsigned" type="Unsigned8"/>
         <xsd:element name="long-unsigned" type="Unsigned16"/>
         <xsd:element name="compact-array">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="contents-description" type="TypeDescription"/>
                  <xsd:element name="array-contents" type="xsd:hexBinary"/>
```

```
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="long64" type="Integer64"/>
    <xsd:element name="long64-unsigned" type="Unsigned64"/>
    <xsd:element name="enum" type="Unsigned8"/>
    <xsd:element name="float32" type="xsd:float"/>
    <xsd:element name="float64" type="xsd:double"/>
    <xsd:element name="date-time">
      <xsd:simpleType>
        <xsd:restriction base="xsd:hexBinary">
          <xsd:length value="12"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="date">
      <xsd:simpleType>
        <xsd:restriction base="xsd:hexBinary">
          <xsd:length value="5"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="time">
      <xsd:simpleType>
        <xsd:restriction base="xsd:hexBinary">
          <xsd:length value="4"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="delta-integer" type="Integer8"/>
    <xsd:element name="delta-long" type="Integer16"/>
    <xsd:element name="delta-double-long" type="Integer32"/>
```

```
            <xsd:element name="delta-unsigned" type="Unsigned8"/>
            <xsd:element name="delta-long-unsigned" type="Unsigned16"/>
            <xsd:element name="delta-double-long-unsigned" type="Unsigned32"/>
            <xsd:element name="dont-care" type="NULL"/>
         </xsd:choice>
      </xsd:complexType>

      <xsd:complexType name="Parameterized-Access">
         <xsd:sequence>
            <xsd:element name="variable-name" type="ObjectName"/>
            <xsd:element name="selector" type="Unsigned8"/>
            <xsd:element name="parameter" type="Data"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Block-Number-Access">
         <xsd:sequence>
            <xsd:element name="block-number" type="Unsigned16"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Read-Data-Block-Access">
         <xsd:sequence>
            <xsd:element name="last-block" type="xsd:boolean"/>
            <xsd:element name="block-number" type="Unsigned16"/>
            <xsd:element name="raw-data" type="xsd:hexBinary"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Write-Data-Block-Access">
         <xsd:sequence>
            <xsd:element name="last-block" type="xsd:boolean"/>
```

```
         <xsd:element name="block-number" type="Unsigned16"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Variable-Access-Specification">
      <xsd:choice>
         <xsd:element name="variable-name" type="ObjectName"/>
         <xsd:element name="parameterized-access" type="Parameterized-Access"/>
         <xsd:element name="block-number-access" type="Block-Number-Access"/>
         <xsd:element name="read-data-block-access" type="Read-Data-Block-Access"/>
         <xsd:element name="write-data-block-access" type="Write-Data-Block-Access"/>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="ReadRequest">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
         <xsd:element name="Variable-Access-Specification" type="Variable-Access-Specification"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="WriteRequest">
      <xsd:sequence>
         <xsd:element name="variable-access-specification">
            <xsd:complexType>
               <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="Variable-Access-Specification" type="Variable-Access-
Specification"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="list-of-data">
            <xsd:complexType>
               <xsd:sequence minOccurs="0" maxOccurs="unbounded">
```

```
                    <xsd:element name="Data" type="Data"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="InitiateResponse">
    <xsd:sequence>
        <xsd:element name="negotiated-quality-of-service" minOccurs="0" type="Integer8"/>
        <xsd:element name="negotiated-dlms-version-number" type="Unsigned8"/>
        <xsd:element name="negotiated-conformance" type="Conformance"/>
        <xsd:element name="server-max-receive-pdu-size" type="Unsigned16"/>
        <xsd:element name="vaa-name" type="ObjectName"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Data-Block-Result">
    <xsd:sequence>
        <xsd:element name="last-block" type="xsd:boolean"/>
        <xsd:element name="block-number" type="Unsigned16"/>
        <xsd:element name="raw-data" type="xsd:hexBinary"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ReadResponse">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="CHOICE">
            <xsd:complexType>
                <xsd:choice>
                    <xsd:element name="data" type="Data"/>
                    <xsd:element name="data-access-error" type="Data-Access-Result"/>
```

```
                  <xsd:element name="data-block-result" type="Data-Block-Result"/>
                  <xsd:element name="block-number" type="Unsigned16"/>
               </xsd:choice>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="WriteResponse">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
         <xsd:element name="CHOICE">
            <xsd:complexType>
               <xsd:choice>
                  <xsd:element name="success" type="NULL"/>
                  <xsd:element name="data-access-error" type="Data-Access-Result"/>
                  <xsd:element name="block-number" type="Unsigned16"/>
               </xsd:choice>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="ServiceError">
      <xsd:choice>
         <xsd:element name="application-reference">
            <xsd:simpleType>
               <xsd:restriction base="xsd:token">
                  <xsd:enumeration value="other"/>
                  <xsd:enumeration value="time-elapsed"/>
                  <xsd:enumeration value="application-unreachable"/>
                  <xsd:enumeration value="application-reference-invalid"/>
                  <xsd:enumeration value="application-context-unsupported"/>
```

```
         <xsd:enumeration value="provider-communication-error"/>
         <xsd:enumeration value="deciphering-error"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="hardware-resource">
   <xsd:simpleType>
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="other"/>
         <xsd:enumeration value="memory-unavailable"/>
         <xsd:enumeration value="processor-resource-unavailable"/>
         <xsd:enumeration value="mass-storage-unavailable"/>
         <xsd:enumeration value="other-resource-unavailable"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="vde-state-error">
   <xsd:simpleType>
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="other"/>
         <xsd:enumeration value="no-dlms-context"/>
         <xsd:enumeration value="loading-data-set"/>
         <xsd:enumeration value="status-nochange"/>
         <xsd:enumeration value="status-inoperable"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="service">
   <xsd:simpleType>
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="other"/>
         <xsd:enumeration value="pdu-size"/>
```

```
         <xsd:enumeration value="service-unsupported"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="definition">
   <xsd:simpleType>
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="other"/>
         <xsd:enumeration value="object-undefined"/>
         <xsd:enumeration value="object-class-inconsistent"/>
         <xsd:enumeration value="object-attribute-inconsistent"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="access">
   <xsd:simpleType>
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="other"/>
         <xsd:enumeration value="scope-of-access-violated"/>
         <xsd:enumeration value="object-access-violated"/>
         <xsd:enumeration value="hardware-fault"/>
         <xsd:enumeration value="object-unavailable"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="initiate">
   <xsd:simpleType>
      <xsd:restriction base="xsd:token">
         <xsd:enumeration value="other"/>
         <xsd:enumeration value="dlms-version-too-low"/>
         <xsd:enumeration value="incompatible-conformance"/>
         <xsd:enumeration value="pdu-size-too-short"/>
```

```
                    <xsd:enumeration value="refused-by-the-VDE-Handler"/>
                 </xsd:restriction>
              </xsd:simpleType>
           </xsd:element>
           <xsd:element name="load-data-set">
              <xsd:simpleType>
                 <xsd:restriction base="xsd:token">
                    <xsd:enumeration value="other"/>
                    <xsd:enumeration value="primitive-out-of-sequence"/>
                    <xsd:enumeration value="not-loadable"/>
                    <xsd:enumeration value="dataset-size-too-large"/>
                    <xsd:enumeration value="not-awaited-segment"/>
                    <xsd:enumeration value="interpretation-failure"/>
                    <xsd:enumeration value="storage-failure"/>
                    <xsd:enumeration value="data-set-not-ready"/>
                 </xsd:restriction>
              </xsd:simpleType>
           </xsd:element>
           <xsd:element name="task">
              <xsd:simpleType>
                 <xsd:restriction base="xsd:token">
                    <xsd:enumeration value="other"/>
                    <xsd:enumeration value="no-remote-control"/>
                    <xsd:enumeration value="ti-stopped"/>
                    <xsd:enumeration value="ti-running"/>
                    <xsd:enumeration value="ti-unusable"/>
                 </xsd:restriction>
              </xsd:simpleType>
           </xsd:element>
        </xsd:choice>
     </xsd:complexType>

```

```
<xsd:complexType name="ConfirmedServiceError">
   <xsd:choice>
      <xsd:element name="initiateError" type="ServiceError"/>
      <xsd:element name="getStatus" type="ServiceError"/>
      <xsd:element name="getNameList" type="ServiceError"/>
      <xsd:element name="getVariableAttribute" type="ServiceError"/>
      <xsd:element name="read" type="ServiceError"/>
      <xsd:element name="write" type="ServiceError"/>
      <xsd:element name="getDataSetAttribute" type="ServiceError"/>
      <xsd:element name="getTIAttribute" type="ServiceError"/>
      <xsd:element name="changeScope" type="ServiceError"/>
      <xsd:element name="start" type="ServiceError"/>
      <xsd:element name="stop" type="ServiceError"/>
      <xsd:element name="resume" type="ServiceError"/>
      <xsd:element name="makeUsable" type="ServiceError"/>
      <xsd:element name="initiateLoad" type="ServiceError"/>
      <xsd:element name="loadSegment" type="ServiceError"/>
      <xsd:element name="terminateLoad" type="ServiceError"/>
      <xsd:element name="initiateUpLoad" type="ServiceError"/>
      <xsd:element name="upLoadSegment" type="ServiceError"/>
      <xsd:element name="terminateUpLoad" type="ServiceError"/>
   </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Notification-Body">
   <xsd:sequence>
      <xsd:element name="data-value" type="Data"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Data-Notification">
   <xsd:sequence>
```

```
        <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
        <xsd:element name="date-time" type="xsd:hexBinary"/>
        <xsd:element name="notification-body" type="Notification-Body"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="UnconfirmedWriteRequest">
      <xsd:sequence>
        <xsd:element name="variable-access-specification">
          <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="Variable-Access-Specification" type="Variable-Access-Specification"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="list-of-data">
          <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="Data" type="Data"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="InformationReportRequest">
      <xsd:sequence>
        <xsd:element name="current-time" minOccurs="0" type="xsd:dateTime"/>
        <xsd:element name="variable-access-specification">
          <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
```

```
                  <xsd:element name="Variable-Access-Specification" type="Variable-Access-
Specification"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="list-of-data">
            <xsd:complexType>
               <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="Data" type="Data"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Cosem-Attribute-Descriptor">
      <xsd:sequence>
         <xsd:element name="class-id" type="Cosem-Class-Id"/>
         <xsd:element name="instance-id" type="Cosem-Object-Instance-Id"/>
         <xsd:element name="attribute-id" type="Cosem-Object-Attribute-Id"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Selective-Access-Descriptor">
      <xsd:sequence>
         <xsd:element name="access-selector" type="Unsigned8"/>
         <xsd:element name="access-parameters" type="Data"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Get-Request-Normal">
      <xsd:sequence>
         <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
```

```
            <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
            <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Get-Request-Next">
         <xsd:sequence>
            <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
            <xsd:element name="block-number" type="Unsigned32"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Cosem-Attribute-Descriptor-With-Selection">
         <xsd:sequence>
            <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
            <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Get-Request-With-List">
         <xsd:sequence>
            <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
            <xsd:element name="attribute-descriptor-list">
               <xsd:complexType>
                  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                     <xsd:element name="Cosem-Attribute-Descriptor-With-Selection" type="Cosem-Attribute-
Descriptor-With-Selection"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>

```

```
<xsd:complexType name="Get-Request">
   <xsd:choice>
      <xsd:element name="get-request-normal" type="Get-Request-Normal"/>
      <xsd:element name="get-request-next" type="Get-Request-Next"/>
      <xsd:element name="get-request-with-list" type="Get-Request-With-List"/>
   </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Set-Request-Normal">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
      <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
      <xsd:element name="value" type="Data"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DataBlock-SA">
   <xsd:sequence>
      <xsd:element name="last-block" type="xsd:boolean"/>
      <xsd:element name="block-number" type="Unsigned32"/>
      <xsd:element name="raw-data" type="xsd:hexBinary"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Request-With-First-Datablock">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
      <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
      <xsd:element name="datablock" type="DataBlock-SA"/>
   </xsd:sequence>
```

```
  </xsd:complexType>

  <xsd:complexType name="Set-Request-With-Datablock">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="datablock" type="DataBlock-SA"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Set-Request-With-List">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="attribute-descriptor-list">
        <xsd:complexType>
          <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="Cosem-Attribute-Descriptor-With-Selection" type="Cosem-Attribute-Descriptor-With-Selection"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="value-list">
        <xsd:complexType>
          <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="Data" type="Data"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Set-Request-With-List-And-First-Datablock">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
```

```
          <xsd:element name="attribute-descriptor-list">
            <xsd:complexType>
              <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="Cosem-Attribute-Descriptor-With-Selection" type="Cosem-Attribute-Descriptor-With-Selection"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="datablock" type="DataBlock-SA"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Set-Request">
        <xsd:choice>
          <xsd:element name="set-request-normal" type="Set-Request-Normal"/>
          <xsd:element name="set-request-with-first-datablock" type="Set-Request-With-First-Datablock"/>
          <xsd:element name="set-request-with-datablock" type="Set-Request-With-Datablock"/>
          <xsd:element name="set-request-with-list" type="Set-Request-With-List"/>
          <xsd:element name="set-request-with-list-and-first-datablock" type="Set-Request-With-List-And-First-Datablock"/>
        </xsd:choice>
      </xsd:complexType>

      <xsd:complexType name="EventNotificationRequest">
        <xsd:sequence>
          <xsd:element name="time" minOccurs="0" type="xsd:hexBinary"/>
          <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
          <xsd:element name="attribute-value" type="Data"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Cosem-Method-Descriptor">
        <xsd:sequence>
```

```
      <xsd:element name="class-id" type="Cosem-Class-Id"/>
      <xsd:element name="instance-id" type="Cosem-Object-Instance-Id"/>
      <xsd:element name="method-id" type="Cosem-Object-Method-Id"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Action-Request-Normal">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="cosem-method-descriptor" type="Cosem-Method-Descriptor"/>
      <xsd:element name="method-invocation-parameters" minOccurs="0" type="Data"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Action-Request-Next-Pblock">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="block-number" type="Unsigned32"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Action-Request-With-List">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="cosem-method-descriptor-list">
        <xsd:complexType>
          <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="Cosem-Method-Descriptor" type="Cosem-Method-Descriptor"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="method-invocation-parameters">
```

```
            <xsd:complexType>
               <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="Data" type="Data"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Action-Request-With-First-Pblock">
      <xsd:sequence>
         <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
         <xsd:element name="cosem-method-descriptor" type="Cosem-Method-Descriptor"/>
         <xsd:element name="pblock" type="DataBlock-SA"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Action-Request-With-List-And-First-Pblock">
      <xsd:sequence>
         <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
         <xsd:element name="cosem-method-descriptor-list">
            <xsd:complexType>
               <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="Cosem-Method-Descriptor" type="Cosem-Method-Descriptor"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="pblock" type="DataBlock-SA"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Action-Request-With-Pblock">
```

```
      <xsd:sequence>
         <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
         <xsd:element name="pblock" type="DataBlock-SA"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Action-Request">
      <xsd:choice>
         <xsd:element name="action-request-normal" type="Action-Request-Normal"/>
         <xsd:element name="action-request-next-pblock" type="Action-Request-Next-Pblock"/>
         <xsd:element name="action-request-with-list" type="Action-Request-With-List"/>
         <xsd:element name="action-request-with-first-pblock" type="Action-Request-With-First-Pblock"/>
         <xsd:element name="action-request-with-list-and-first-pblock" type="Action-Request-With-List-And-First-Pblock"/>
         <xsd:element name="action-request-with-pblock" type="Action-Request-With-Pblock"/>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="Get-Data-Result">
      <xsd:choice>
         <xsd:element name="data" type="Data"/>
         <xsd:element name="data-access-result" type="Data-Access-Result"/>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="Get-Response-Normal">
      <xsd:sequence>
         <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
         <xsd:element name="result" type="Get-Data-Result"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="DataBlock-G">
```

```
        <xsd:sequence>
            <xsd:element name="last-block" type="xsd:boolean"/>
            <xsd:element name="block-number" type="Unsigned32"/>
            <xsd:element name="result">
                <xsd:complexType>
                    <xsd:choice>
                        <xsd:element name="raw-data" type="xsd:hexBinary"/>
                        <xsd:element name="data-access-result" type="Data-Access-Result"/>
                    </xsd:choice>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Get-Response-With-Datablock">
        <xsd:sequence>
            <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
            <xsd:element name="result" type="DataBlock-G"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Get-Response-With-List">
        <xsd:sequence>
            <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
            <xsd:element name="result">
                <xsd:complexType>
                    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="Get-Data-Result" type="Get-Data-Result"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
```

```
</xsd:complexType>

<xsd:complexType name="Get-Response">
   <xsd:choice>
      <xsd:element name="get-response-normal" type="Get-Response-Normal"/>
      <xsd:element name="get-response-with-datablock" type="Get-Response-With-Datablock"/>
      <xsd:element name="get-response-with-list" type="Get-Response-With-List"/>
   </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Set-Response-Normal">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="result" type="Data-Access-Result"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-Datablock">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="block-number" type="Unsigned32"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-Last-Datablock">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="result" type="Data-Access-Result"/>
      <xsd:element name="block-number" type="Unsigned32"/>
   </xsd:sequence>
</xsd:complexType>

```

```
<xsd:complexType name="Set-Response-Last-Datablock-With-List">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="result">
         <xsd:simpleType>
            <xsd:list itemType="Data-Access-Result"/>
         </xsd:simpleType>
      </xsd:element>
      <xsd:element name="block-number" type="Unsigned32"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-With-List">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="result">
         <xsd:simpleType>
            <xsd:list itemType="Data-Access-Result"/>
         </xsd:simpleType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response">
   <xsd:choice>
      <xsd:element name="set-response-normal" type="Set-Response-Normal"/>
      <xsd:element name="set-response-datablock" type="Set-Response-Datablock"/>
      <xsd:element name="set-response-last-datablock" type="Set-Response-Last-Datablock"/>
      <xsd:element name="set-response-last-datablock-with-list" type="Set-Response-Last-Datablock-
With-List"/>
      <xsd:element name="set-response-with-list" type="Set-Response-With-List"/>
   </xsd:choice>
</xsd:complexType>
```

```
<xsd:complexType name="Action-Response-With-Optional-Data">
   <xsd:sequence>
      <xsd:element name="result" type="Action-Result"/>
      <xsd:element name="return-parameters" minOccurs="0" type="Get-Data-Result"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-Normal">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="single-response" type="Action-Response-With-Optional-Data"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-With-Pblock">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="pblock" type="DataBlock-SA"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-With-List">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="list-of-responses">
         <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="Action-Response-With-Optional-Data" type="Action-Response-With-Optional-Data"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
```

```
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Action-Response-Next-Pblock">
    <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="block-number" type="Unsigned32"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Action-Response">
    <xsd:choice>
      <xsd:element name="action-response-normal" type="Action-Response-Normal"/>
      <xsd:element name="action-response-with-pblock" type="Action-Response-With-Pblock"/>
      <xsd:element name="action-response-with-list" type="Action-Response-With-List"/>
      <xsd:element name="action-response-next-pblock" type="Action-Response-Next-Pblock"/>
    </xsd:choice>
  </xsd:complexType>

  <xsd:complexType name="ExceptionResponse">
    <xsd:sequence>
      <xsd:element name="state-error">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="service-not-allowed"/>
            <xsd:enumeration value="service-unknown"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="service-error">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
```

```
               <xsd:enumeration value="operation-not-possible"/>
               <xsd:enumeration value="service-not-supported"/>
               <xsd:enumeration value="other-reason"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Get">
   <xsd:sequence>
      <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Set">
   <xsd:sequence>
      <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Action">
   <xsd:sequence>
      <xsd:element name="cosem-method-descriptor" type="Cosem-Method-Descriptor"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Get-With-Selection">
   <xsd:sequence>
      <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
      <xsd:element name="access-selection" type="Selective-Access-Descriptor"/>
   </xsd:sequence>
```

```
    </xsd:complexType>

    <xsd:complexType name="Access-Request-Set-With-Selection">
       <xsd:sequence>
          <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
          <xsd:element name="access-selection" type="Selective-Access-Descriptor"/>
       </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Access-Request-Specification">
       <xsd:choice>
          <xsd:element name="access-request-get" type="Access-Request-Get"/>
          <xsd:element name="access-request-set" type="Access-Request-Set"/>
          <xsd:element name="access-request-action" type="Access-Request-Action"/>
          <xsd:element name="access-request-get-with-selection" type="Access-Request-Get-With-
Selection"/>
          <xsd:element name="access-request-set-with-selection" type="Access-Request-Set-With-
Selection"/>
       </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="List-Of-Access-Request-Specification">
       <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Access-Request-Specification" type="Access-Request-Specification"/>
       </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="List-Of-Data">
       <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Data" type="Data"/>
       </xsd:sequence>
    </xsd:complexType>
```

```
<xsd:complexType name="Access-Request-Body">
   <xsd:sequence>
      <xsd:element name="access-request-specification" type="List-Of-Access-Request-Specification"/>
      <xsd:element name="access-request-list-of-data" type="List-Of-Data"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request">
   <xsd:sequence>
      <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
      <xsd:element name="date-time" type="xsd:hexBinary"/>
      <xsd:element name="access-request-body" type="Access-Request-Body"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Get">
   <xsd:sequence>
      <xsd:element name="result" type="Data-Access-Result"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Set">
   <xsd:sequence>
      <xsd:element name="result" type="Data-Access-Result"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Action">
   <xsd:sequence>
      <xsd:element name="result" type="Action-Result"/>
   </xsd:sequence>
</xsd:complexType>
```

```
    <xsd:complexType name="Access-Response-Specification">
        <xsd:choice>
            <xsd:element name="access-response-get" type="Access-Response-Get"/>
            <xsd:element name="access-response-set" type="Access-Response-Set"/>
            <xsd:element name="access-response-action" type="Access-Response-Action"/>
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="List-Of-Access-Response-Specification">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="Access-Response-Specification" type="Access-Response-Specification"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Access-Response-Body">
        <xsd:sequence>
            <xsd:element name="access-request-specification" minOccurs="0" type="List-Of-Access-Request-Specification"/>
            <xsd:element name="access-response-list-of-data" type="List-Of-Data"/>
            <xsd:element name="access-response-specification" type="List-Of-Access-Response-Specification"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Access-Response">
        <xsd:sequence>
            <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
            <xsd:element name="date-time" type="xsd:hexBinary"/>
            <xsd:element name="access-response-body" type="Access-Response-Body"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
<xsd:complexType name="General-Glo-Ciphering">
   <xsd:sequence>
      <xsd:element name="system-title" type="xsd:hexBinary"/>
      <xsd:element name="ciphered-content" type="xsd:hexBinary"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="General-Ded-Ciphering">
   <xsd:sequence>
      <xsd:element name="system-title" type="xsd:hexBinary"/>
      <xsd:element name="ciphered-content" type="xsd:hexBinary"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Identified-Key">
   <xsd:sequence>
      <xsd:element name="key-id" type="Key-Id"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Wrapped-Key">
   <xsd:sequence>
      <xsd:element name="kek-id" type="Kek-Id"/>
      <xsd:element name="key-ciphered-data" type="xsd:hexBinary"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Agreed-Key">
   <xsd:sequence>
      <xsd:element name="key-parameters" type="xsd:hexBinary"/>
      <xsd:element name="key-ciphered-data" type="xsd:hexBinary"/>
   </xsd:sequence>
```

```
</xsd:complexType>

<xsd:complexType name="Key-Info">
   <xsd:choice>
      <xsd:element name="identified-key" type="Identified-Key"/>
      <xsd:element name="wrapped-key" type="Wrapped-Key"/>
      <xsd:element name="agreed-key" type="Agreed-Key"/>
   </xsd:choice>
</xsd:complexType>

<xsd:complexType name="General-Ciphering">
   <xsd:sequence>
      <xsd:element name="transaction-id" type="xsd:hexBinary"/>
      <xsd:element name="originator-system-title" type="xsd:hexBinary"/>
      <xsd:element name="recipient-system-title" type="xsd:hexBinary"/>
      <xsd:element name="date-time" type="xsd:hexBinary"/>
      <xsd:element name="other-information" type="xsd:hexBinary"/>
      <xsd:element name="key-info" minOccurs="0" type="Key-Info"/>
      <xsd:element name="ciphered-content" type="xsd:hexBinary"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="General-Signing">
   <xsd:sequence>
      <xsd:element name="transaction-id" type="xsd:hexBinary"/>
      <xsd:element name="originator-system-title" type="xsd:hexBinary"/>
      <xsd:element name="recipient-system-title" type="xsd:hexBinary"/>
      <xsd:element name="date-time" type="xsd:hexBinary"/>
      <xsd:element name="other-information" type="xsd:hexBinary"/>
      <xsd:element name="content" type="xsd:hexBinary"/>
      <xsd:element name="signature" type="xsd:hexBinary"/>
   </xsd:sequence>
```

```
   </xsd:complexType>

   <xsd:complexType name="General-Block-Transfer">
      <xsd:sequence>
         <xsd:element name="block-control" type="Block-Control"/>
         <xsd:element name="block-number" type="Unsigned16"/>
         <xsd:element name="block-number-ack" type="Unsigned16"/>
         <xsd:element name="block-data" type="xsd:hexBinary"/>
      </xsd:sequence>
   </xsd:complexType>

</xsd:schema>
```

Annex A
(normative)

Using the DLMS/COSEM application layer in various communications profiles

A.1 General

The COSEM interface model for energy metering equipment, specified in IEC 62056-6-2:2021 has been designed for use with a variety of communication profiles for exchanging data over various communication media. In each such profile, the application layer is the COSEM AL, providing the xDLMS services to access attributes and methods of COSEM objects. For each communication profile, the following elements shall be specified:

- the targeted communication environments;
- the structure of the profile (the set of protocol layers);
- the identification / addressing scheme;
- mapping of the DLMS/COSEM AL services to the service set provided and used by the supporting layer;
- communication profile specific parameters of the DLMS®/COSEM AL services;
- other specific considerations / constraints for using certain services within a given profile.

A.2 Targeted communication environments

This part identifies the communication environments, for which the given communication profile is specified.

A.3 The structure of the profile

This part specifies the protocol layers included in the given profile.

A.4 Identification and addressing schemes

This part describes the identification and addressing schemes specific for the profile.

As described in IEC 62056-6-2:2021 4.1.7, metering equipment are modelled in COSEM as physical devices, containing one or more logical devices. In the COSEM client/server type model, data exchange takes place within application associations, between a COSEM client AP and a COSEM Logical Device, playing the role of a server AP.

To be able to establish the required AA and then to exchange data with the help of the supporting lower layer protocols, the client- and server APs shall be identified and addressed, according to the rules of a communication profile. At least the following elements need to be identified / addressed:

- physical devices hosting clients and servers;
- client- and server APs;

The client- and server APs also identify the AAs.

A.5 Supporting layer services and service mapping

This part specifies the service mapping between the services requested by the COSEM AL and the services provided by its supporting layer.

In each communication profile, the COSEM AL provides the same set of services to the client- and server APs. However, the supporting protocol layer in the various profiles provides a different set of services to the service user AL.

The service mapping specifies how the AL is using the services of its supporting layer to provide ACSE and xDLMS services to its service user. For this purpose, MSCs are generally used, showing the sequence of the events following a service invocation by the COSEM AP.

A.6 Communication profile specific parameters of the COSEM AL services

In COSEM, only the COSEM-OPEN service has communication profile specific parameters (the Protocol_Connection_Parameters). Their values and use are defined as part of the communication profile specification.

A.7 Specific considerations / constraints using certain services within a given profile

The availability and the protocol of some of the services may depend on the communication profile. These elements are specified as part of the communication profile specification.

A.8 The 3-layer, connection-oriented, HDLC based communication profile

This profile is specified in IEC 62056-7-6.

A.9 The TCP-UDP/IP based communication profiles (COSEM_on_IP)

This profile is specified in IEC 62056-9-7.

A.10 The wired and wireless M-Bus communication profiles

This profiles are specified in IEC 62056-7-3:—..

A.11 The S-FSK PLC profile

This profile is specified in IEC 62056-8-3.

Annex B
(normative)

SMS short wrapper

This Annex specifies the transport of xDLMS APDUs in an SMS.

The payload of an SMS message is the xDLMS APDU prepended with the identifier of the Destination_AP and the Source_AP as shown in Figure B.1:

8 bits	8 bits	N*8bits
Dst_AP	Src_AP	Application layer payload (xDLMS APDU)

- Dst_AP = Destination AP identifies the destination Application Process;
- Src_AP = Source AP identifies the source Application Process..

Figure B.1 – Short wrapper

Table B. 1 specifies the identifiers of reserved Application Processes:

Table B.1 – Reserved Application Processes

Client side reserved SAPs	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
Open for client AP assignment	0x02 …0x0F
	0x11 and up
Server side reserved SAPs	
No-station	0x00
Management Logical Device	0x01
Reserved	0x02..0xF
Open for server SAP assignment	0x10…0x7E
All-station (Broadcast)	0x7F

Annex C
(normative)

Gateway protocol

C.1 General

This Annex specifies a method for exchanging data between DLMS®/COSEM clients and servers via a gateway, when this gateway is connected to a Wide Area Network (WAN) or to a Neighbourhood Network (NN) on the one hand, and to a Local Network (LAN) on the other hand, with DLMS®/COSEM servers connected to this LAN.

The gateway acts bidirectional, i.e. it is also possible for a server in the LAN to send messages to a client in the WAN/NN using the gateway (push application).

The gateway function itself may be implemented in a DLMS®/COSEM meter or in a stand-alone device.

The DLMS®/COSEM specification for meter data exchange is based on the client/server model, where the head end system (HES) acts as a client requesting services, and the end devices (e.g. meters) act as servers providing the services requested. In many cases, the client can reach each meter directly, using unicast, multicast or broadcast messages.

There are cases however, when it is practical to connect several end devices to a LAN, and reach those devices via a gateway. The protocol stack used on the LAN may be the same as the one used on the WAN/NN or it may be different.

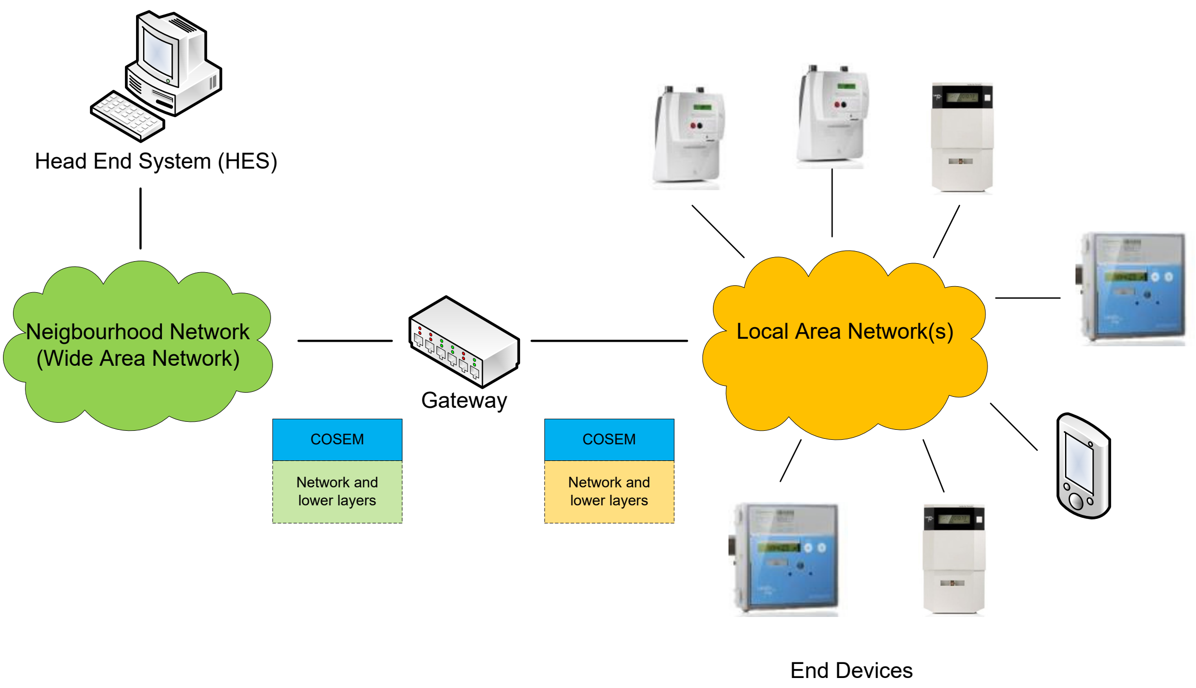


Figure C.1 – General architecture with gateway

The DLMS®/COSEM client (HES) reaches the gateway via a WAN or via NNAP; see Figure C.1. The gateway itself may be a stand-alone device or a DLMS®/COSEM meter capable of acting as a gateway. If configured accordingly, it passes the COSEM APDUs transparently between the HES or NNAP and the COSEM servers (end devices).

The gateway may act bidirectional, i.e. it is also possible for a COSEM server (end device) in the LAN to send COSEM APDUs to the COSEM client (HES) in the WAN/NN using the gateway (push application).

C.2 The gateway protocol

The top layer of any DLMS®/COSEM communication profile is the DLMS®/COSEM application layer.

In order to leave both, the suite of lower layers and the DLMS®/COSEM AL unaffected, the task of routing each message from the client to the end device on the LAN is solved by pre-fixing the COSEM APDUs with a few bytes specifying the network to be used and the address of the device on the LAN and vice versa.

The gateway extracts the payload, a COSEM APDU – together with the application addresses – from the WAN/NN protocol and puts it as a payload to the LAN protocol, and the other way round.

The structure of the prefix with four fields is shown in Figure C.2.

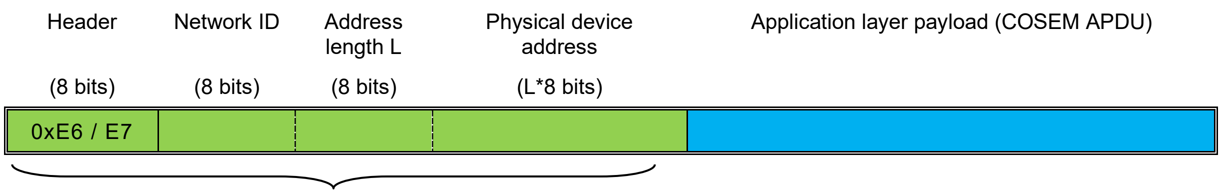


Figure C.2 – The fields used for pre-fixing the COSEM APDUs

- the value of the first byte (header) is either 0xE6 or 0xE7. It indicates that the following bytes don't contain a plain COSEM APDU but contain a COSEM APDU with a prefix:
  - 0xE6 indicates a request message from a DLMS®/COSEM client to a DLMS®/COSEM server or a request message from a DLMS®/COSEM server to a DLMS®/COSEM client (data notification);
  - 0xE7 indicates a response from the DLMS®/COSEM server to the DLMS®/COSEM client;
- the second byte carries an identifier of the destination network where the messages are transferred to. This allows accessing several networks using the same or different communication protocols through the same gateway. The network ID is not linked to the communication protocol and can be set to any value. If only one network exists, 0x00 shall be used.
- the third byte defines the length of the physical device address given in the next L bytes. It depends on the communication protocol used.
- bytes 4 to 4+(L-1) carry the physical device address of the end device or the HES as requested by the communication protocol.

When a telegram with pre-fixed fields reaches a device, which is not a gateway or which doesn't support pre-fixed fields, it shall be simply discarded.

When the client exchanges data directly with the master meter, the pre-fixed fields are not present.

C.3 HES in the WAN/NN acting as Initiator (Pull operation)

In the sequence diagram shown in Figure C.3 the traditional pull data exchange between the DLMS®/COSEM client and server via a gateway is further elaborated:

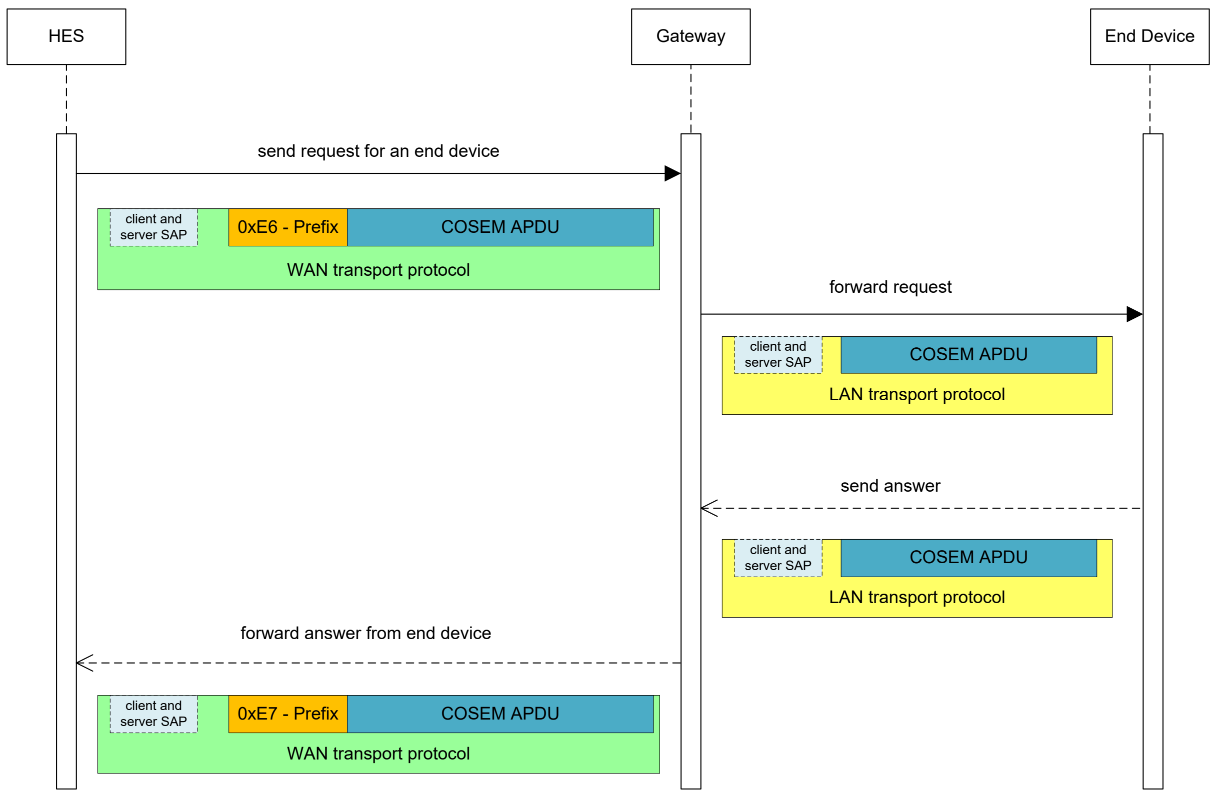


Figure C.3 – Pull message sequence chart

Prerequisite: The client in the WAN/NN has to know the network ID, the protocol and the physical device address of the server it wants to reach in the LAN.

The DLMS®/COSEM client (HES) sends every request, carried by a COSEM APDU, prefixed with four fields as shown in Figure C.2 using the protocol layer supporting the DLMS®/COSEM AL on the WAN/NN.

The gateway forwards each COSEM APDU carrying a .request service primitive to the appropriate network using the network ID and the physical device address contained in the pre-fixed fields. The client and server SAPs are extracted from the protocol layer supporting the DLMS®/COSEM AL on the WAN/NN and inserted into the supporting layer of the DLMS®/COSEM AL on the LAN.

The APDUs carrying the requests do not have any prefix when they arrive in the end devices in the LAN. Every end device processes the request and provides the answer the same way as if it's connected directly to the client.

When the device responds to a request, it is done as if it's connected to the client directly: The APDU does not need to be pre-fixed.

When the gateway receives a COSEM APDU carrying a .response service primitive on the LAN, it extracts the client and server SAPs from the protocol layer supporting the DLMS®/COSEM AL on the LAN. Afterwards it inserts them into the supporting layer of the DLMS®/COSEM AL on the WAN/NN and sends the message with pre-fixed fields to the client using the WAN/NN protocol.

C.4 End devices in the LAN acting as Initiators (Push operation)

C.4.1 General

It is also possible for a server (end device) in the LAN to send messages to a client (HES) in the WAN / NN using the gateway without having received a request service before (push application). Depending on the capabilities of the gateway two scenarios are supported.

C.4.2 End device with WAN/NN knowledge

Prerequisite: The server in the LAN has to know the network ID, the protocol and the address of the client it wants to reach in the WAN/NN.

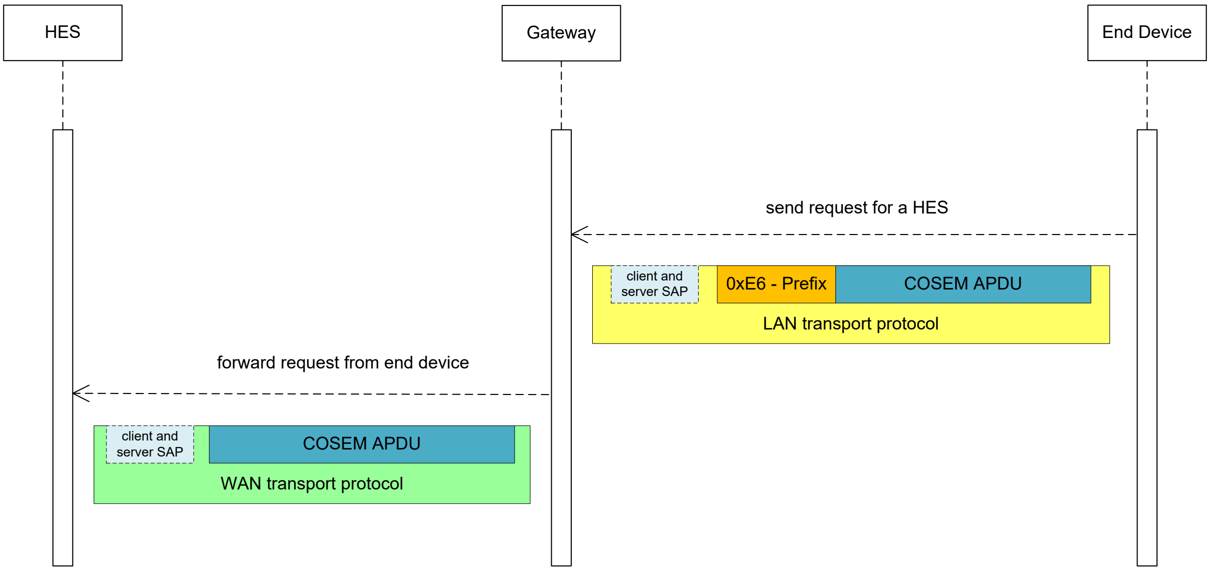


Figure C.4 – Push message sequence chart

The server (end device) sends every request (e.g. data notification request), carried by the COSEM APDU, pre-fixed with 4 fields as defined before to the gateway using the protocol layer supporting the DLMS®/COSEM AL on the LAN, as shown in Figure C.4.

The gateway forwards each COSEM APDU carrying a .request service primitive using the network ID, the protocol and the client address (e.g. WAN/NN MAC address) contained in the pre-fixed fields.

The client and server SAPs are extracted from the protocol layer supporting the DLMS®/COSEM AL on the LAN and inserted into the supporting layer of the DLMS®/COSEM AL on the WAN/NN.

C.4.3 End devices without WAN/NN knowledge

If the end device has no knowledge on the WAN/NN network or if it has no knowledge if it is connected to a gateway at all, it can send standard (not pre-fixed) data notification requests to the gateway. It is then the duty of the gateway to further deal with such messages.

Since this does not require a protocol extension, this use case is not described any further.

C.5 Security

The DLMS®/COSEM AL security mechanisms ensure end-to-end security through the gateway.

Annex D
(informative)

AARQ and AARE encoding examples

D.1 General

This annex contains examples of encoding of the AARQ and AARE APDUs, in cases of using various levels of authentication and in cases of success and failure.

The AARQ, AARE, RLRQ and RLRE APDUs – see 7.2 – shall be encoded in BER (ISO/IEC 8825-1). The user-information field of the AARQ and AARE APDUs contains the xDLMS InitiateRequest / InitiateResponse or ConfirmedServiceError APDUs respectively, encoded in A-XDR as OCTET STRING.

D.2 Encoding of the xDLMS InitiateRequest / InitiateResponse APDU

The xDLMS InitiateRequest / InitiateResponse APDUs are specified as follows:

```
InitiateRequest::= SEQUENCE
{
--  shall not be encoded in DLMS without ciphering
    dedicated-key                       OCTET STRING OPTIONAL,
    response-allowed                    BOOLEAN DEFAULT TRUE,
    proposed-quality-of-service         IMPLICIT Integer8 OPTIONAL,
    proposed-dlms-version-number        Unsigned8,
    proposed-conformance                Conformance,
    client-max-receive-pdu-size         Unsigned16
}

InitiateResponse::= SEQUENCE
{
    negotiated-quality-of-service       IMPLICIT Integer8 OPTIONAL,
    negotiated-dlms-version-number      Unsigned8,
    negotiated-conformance              Conformance,
    server-max-receive-pdu-size         Unsigned16,
    vaa-name                            ObjectName
}
```

The xDLMS InitiateRequest and InitiateResponse APDUs are encoded in A-XDR and they are inserted in the user-information field of the AARQ / AARE APDU respectively.

In the examples below, the following values are used:

- dedicated key: not present; no ciphering is used;
- response-allowed: TRUE (default value);
- proposed-quality-of-service and negotiated-quality-of-service: not present (not used in DLMS®/COSEM);
- proposed-conformance and negotiated-conformance: see below;
- proposed-dlms-version-number and negotiated-dlms-version-number = 6;
- client-max-receive-pdu-size: 1200_D = 0x04B0;
- server-max-receive-pdu-size: 500_D = 0x01F4;
- vaa-name in the case of LN referencing: the dummy value 0x0007;
- vaa-name in the case of SN referencing: the base_name of the current “Association SN” object, 0xFA00.
- the proposed-conformance and the negotiated-conformance elements carry the proposed conformance block and the negotiated conformance block respectively. The values of these examples, for LN referencing and SN referencing respectively, are shown in Table D. 1.

Table D.1 – Conformance block

Conformance::= **[APPLICATION 31] IMPLICIT BIT STRING** (SIZE(24))			**LN referencing**		**SN referencing**	
-- the bit is set when the corresponding service or functionality is available		**Used with**	**Proposed/**	**Negotiated**	**Proposed /**	**Negotiated**
reserved-zero	(0),		0	0	0	0
reserved-one	(1),		0	0	0	0
reserved-two	(2),		0	0	0	0
read	(3),	SN	0	0	1	1
write	(4),	SN	0	0	1	1
unconfirmed-write	(5),	SN	0	0	1	1
reserved-six	(6),		0	0	0	0
reserved-seven	(7),		0	0	0	0
attribute0-supported-with-set	(8),	LN	0	0	0	0
priority-mgmt-supported	(9),	LN	1	1	0	0
attribute0-supported-with-get	(10),	LN	1	0	0	0
block-transfer-with-get-or-read	(11),	LN	1	1	0	0
block-transfer-with-set-or-write	(12),	LN	1	0	0	0
block-transfer-with-action	(13),	LN	1	0	0	0
multiple-references	(14),	LN/SN	1	0	1	1
information-report	(15),	SN	0	0	1	1
reserved-sixteen	(16),		0	0	0	0
reserved-seventeen	(17),		0	0	0	0
parameterized-access	(18),	SN	0	0	1	1
get	(19),	LN	1	1	0	0
set	(20),	LN	1	1	0	0
selective-access	(21),	LN	1	1	0	0
event-notification	(22),	LN	1	1	0	0
action	(23)	LN	1	1	0	0
Value of the bit string			00 7E 1F	00 50 1F	1C 03 20	1C 03 20

With these parameters, the A-XDR encoding of the xDLMS InitiateRequest APDU is as shown in Table D.2.

Table D.2 – A-XDR encoding of the xDLMS InitiateRequest APDU

-- A-XDR encoding of the xDLMS InitiateRequest APDU	**LN referencing**	**SN referencing**
// encoding of the tag of the DLMS APDU CHOICE *(InitiateRequest)*	01	01
*-- encoding of the dedicated-key component (**OCTET STRING OPTIONAL**)*		
// usage flag*(**FALSE**, not present)*	00	00
*-- encoding of the response-allowed component (**BOOLEAN DEFAULT TRUE**)*		
// usage flag*(**FALSE**, default value **TRUE** conveyed)*	00	00
*-- encoding of the proposed-quality-of-service component ([0] **IMPLICIT** Integer8 **OPTIONAL**)*		
// usage flag*(**FALSE**, not present)*	00	00
-- encoding of the proposed-dlms-version-number component (Unsigned8)		
// value= 6, the encoding of an Unsigned8 is its value	06	06
*-- encoding of the proposed-conformance component (Conformance, [APPLICATION 31] **IMPLICIT BIT STRING** (SIZE(24))* [1]		
// encoding of the [APPLICATION 31] tag *(ASN.1 explicit tag)* [2]	5F 1F	5F 1F
// encoding of the length of the 'contents' field in octet *(4)*	04	04
// encoding of the number of unused bits in the final octet of the BIT STRING *(0)*	00	00
// encoding of the fixed length BIT STRING value	00 7E 1F	1C 03 20
-- encoding of the client-max-receive-pdu-size component (Unsigned16)		
// value = 0x04B0, the encoding of an Unsigned16 is its value	04 B0	04 B0
-- resulting octet-string, to be inserted in the user-information field of the AARQ APDU	01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0	01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0

[1] As specified in IEC 61334-6:2000, Annex C, Examples 1 and 2, the proposed-conformance element of the xDLMS InitiateRequest APDU and the negotiated-conformance element of the xDLMS InitiateResponse APDU are encoded in BER. That's why the length of the bit-string and the number of the unused bits are encoded.

[2] For encoding of identifier octets see ISO/IEC 8825-1:2008, 8.1.2. For compliance with existing implementations, encoding of the [Application 31] tag on one byte (5F) instead of two bytes (5F 1F) is accepted when the 3-layer, connection-oriented, HDLC-based profile is used.

The A-XDR encoding of the xDLMS InitiateResponse APDU is as shown in Table D.3

Table D.3 – A-XDR encoding of the xDLMS InitiateResponse APDU

-- A-XDR encoding of the xDLMS InitiateResponse APDU	**LN referencing**	**SN referencing**
// encoding of the tag of the DLMS APDU CHOICE *(InitiateResponse)*	08	08
-- encoding of the negotiated-quality-of-service component ([0] ***IMPLICIT*** *Integer8* ***OPTIONAL****)*		
// usage flag*(**FALSE**, not present)*	00	00
-- encoding of the negotiated-dlms-version-number component (Unsigned8)		
// value = 6, the encoding of an Unsigned8 is its value	06	06
-- encoding of the negotiated-conformance component (Conformance, [APPLICATION 31] ***IMPLICIT BIT STRING*** *(SIZE(24))*		
// encoding of the [APPLICATION 31] tag *(ASN.1 explicit tag)*	5F 1F	5F 1F
// encoding of the length of the 'contents' field in octet *(4)*	04	04
// encoding of the number of unused bits in the final octet of the BIT STRING *(0)*	00	00
// encoding of the fixed length BIT STRING value	00 50 1F	1C 03 20
-- encoding of the server-max-receive-pdu-size component (Unsigned16)		
// value = 0x01F4, the encoding of an Unsigned16 is its value	01 F4	01 F4
-- encoding of the VAA-Name component (ObjectName, Integer16)		
// value=0x0007 for LN and 0xFA00 for SN referencing; the encoding of a value constrained Integer16 is its value	00 07	FA 00
-- resulting octet-string, to be inserted in the user-information field of the AARE APDU	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00

D.3 Specification of the AARQ and AARE APDUs

The AARQ and the AARE APDUs are specified in Clause 8 as follows:

```
AARQ-apdu::= [APPLICATION 0] IMPLICIT SEQUENCE
{
-- [APPLICATION 0] == [ 60H ] = [ 96 ]

   protocol-version                 [0] IMPLICIT    BIT STRING {version1 (0)}
                                                    DEFAULT {version1},
   application-context-name         [1]             Application-context-name,
   called-AP-title                  [2]             AP-title OPTIONAL,
   called-AE-qualifier              [3]             AE-qualifier OPTIONAL,
   called-AP-invocation-id          [4]             AP-invocation-identifier OPTIONAL,
   called-AE-invocation-id          [5]             AE-invocation-identifier OPTIONAL,
   calling-AP-title                 [6]             AP-title OPTIONAL,
   calling-AE-qualifier             [7]             AE-qualifier OPTIONAL,
   calling-AP-invocation-id         [8]             AP-invocation-identifier OPTIONAL,
   calling-AE-invocation-id         [9]             AE-invocation-identifier OPTIONAL,

-- The following field shall not be present if only the kernel is used.
```

```
   sender-acse-requirements          [10] IMPLICIT   ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
-- selected.
   mechanism-name                    [11] IMPLICIT   Mechanism-name OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
--selected.
   calling-authentication-value      [12] EXPLICIT   Authentication-value OPTIONAL,
    implementation-information       [29] IMPLICIT   Implementation-data OPTIONAL,
    user-information                 [30] EXPLICIT   Association-information OPTIONAL
}

-- The user-information field shall carry an InitiateRequest APDU encoded in A-XDR, and
-- then encoding the resulting OCTET STRING in BER.

AARE-apdu::= [APPLICATION 1] IMPLICIT SEQUENCE
{
-- [APPLICATION 1] == [ 61H ] = [ 97 ]

   protocol-version                  [0] IMPLICIT    BIT STRING {version1 (0)}
                                                     DEFAULT {version1},
   application-context-name          [1]             Application-context-name,
   result                            [2]             Association-result,
   result-source-diagnostic          [3]             Associate-source-diagnostic,
   responding-AP-title               [4]             AP-title OPTIONAL,
   responding-AE-qualifier           [5]             AE-qualifier OPTIONAL,
   responding-AP-invocation-id       [6]             AP-invocation-identifier OPTIONAL,
    responding-AE-invocation-id      [7]             AE-invocation-identifier OPTIONAL,

-- The following field shall not be present if only the kernel is used.
   responder-acse-requirements       [8] IMPLICIT    ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
-- selected.
   mechanism-name                    [9] IMPLICIT    Mechanism-name OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
-- selected.
   responding-authentication-value   [10] EXPLICIT   Authentication-value OPTIONAL,
   implementation-information        [29] IMPLICIT   Implementation-data OPTIONAL,
   user-information                  [30] EXPLICIT   Association-information OPTIONAL
}

-- The user-information field shall carry either an InitiateResponse (or, when the
-- proposed xDLMS context is not accepted by the server, a ConfirmedServiceError) APDU
-- encoded in A-XDR. The resulting OCTET STRING shall be encoded in BER.
```

D.4 Data for the examples

In these examples:

- the protocol-version is the default ACSE version;
- the value of the application-context-name:
  - in the case of LN referencing, with no ciphering: 2, 16, 756, 5, 8, 1, 1;
  - in the case of SN referencing, with no ciphering: 2, 16, 756, 5, 8, 1, 2;

- the optional called-AP-title, called-AE-qualifier, called-AP-invocation-id, called-AE-invocation-id, calling-AP-title, calling-AE-qualifier, calling-AP-invocation-id, calling-AE-invocation-id fields of the AARQ, and the optional responding-AP-title, responding-AE-qualifier, responding-AP-invocation-id, responding-AE-invocation-id fields of the AARE are not present;
- the value of the mechanism-name:
  - in the case of low-level-security: 2, 16, 756, 5, 8, 2, 1;
  - in the case of high-level-security (5): 2, 16, 756, 5, 8, 2, 5;
- the calling-authentication-value:
  - in the case of low-level-security is 12345678 (encoded as 31 32 33 34 35 36 37 38);
  - in the case of high-level security, (challenge CtoS) is K56iVagY (encoded as 4B 35 36 69 56 61 67 59);
- the responding authentication-value (challenge StoC) is P6wRJ21F (encoded as 50 36 77 52 4A 32 31 46);
- the optional implementation-information field in the AARQ and AARE APDUs is not present;
- the user-information field carries the xDLMS InitiateRequest / InitiateResponse APDUs as shown above.

The application-context-name and the (authentication) mechanism-name OBJECT IDENTIFIERS are encoded as follows:

- BER Encoding for OBJECT IDENTIFIER is a packed sequence of numbers representing the arc labels. Each number – except the first two, which are combined into one – is represented as a series of octets, with 7 bits being used from each octet and the most significant bit is set to 1 in all but the last octet. The fewest possible number of octets shall be used;
- in the case of the application context name LN referencing with no ciphering, the arc labels of the object identifier are (2, 16, 756, 5, 8, 1, 1);
  - the first octet of the encoding is the combination of the first two numbers into a single number, following the rule of 40*First+Second -> 40*2 + 16 = 96 = 0x60;
  - the third number of the Object Identifier (756) requires two octets: its hexadecimal value is 0x02F4, which is 00000010 11110100, but following the above rule, the MSB of the first octet shall be set to 1 and the MSB of the second (last) octet shall be set to 0, thus this bit shall be shifted into the LSB of the first octet. This gives binary 10000101 01110100, which is 0x8574;
  - each remaining numbers of the Object Identifier required to be encoded on one octet;
  - this results in the encoding 60 85 74 05 08 01 01.
- similarly, in the case of application context name SN referencing with no ciphering the BER encoding is 60 85 74 05 08 01 02;
- in the case of mechanism name low-level-security, the BER encoding is 60 85 74 05 08 02 01;
- in the case of mechanism name high-level-security (5), the BER encoding is 60 85 74 05 08 02 05.

D.5 Encoding of the AARQ APDU

Here, six different cases are shown:

- LN referencing with no ciphering, no security, LLS and HLS;
- SN referencing with no ciphering, no security, LLS and HLS;

The encoding is shown in Table D.4. See also Table D.5.

9202 **Table D.4 – BER encoding of the AARQ APDU**

-- BER encoding of the AARQ APDU	**LN referencing**			**SN referencing**		
	no sec.	**LLS**	**HLS**	**no sec.**	**LLS**	**HLS**
// encoding of the tag of the AARQ APDU *([APPLICATION 0], Application)*	60			60		
// encoding of the length of the AARQ's content's field	1D	36	36	1D	36	36
-- protocol-version field ([0], ***IMPLICIT BIT STRING*** *{ version1 (0) }* ***DEFAULT*** *{ version1 }*						
// no encoding, thus it is considered with its **DEFAULT** value						
-- encoding of the fields of the Kernel						
-- application-context-name field ([1], Application-context-name, ***OBJECT IDENTIFIER****)*						
// encoding of the tag *([1], Context-specific)*	A1			A1		
// encoding of the length of the tagged component's value field	09			09		
// encoding of the choice for application-context-name *(**OBJECT IDENTIFIER**, Universal)*	06			06		
// encoding of the length of the Object Identifier's value field	07			07		
// encoding of the value of the Object Identifier	60 85 74 05 08 01 01			60 85 74 05 08 01 02		
-- encoding of the fields of the authentication functional unit						
--sender-acse-requirements field ([10], ACSE-requirements, ***BIT STRING*** *{ authentication (0) })*						
// encoding of the tag of the acse-requirements field *([10],* ***IMPLICIT****, Context-specific)*	–	8A	8A	–	8A	8A
// encoding of the length of the tagged component's value field	–	02	02	–	02	02
// encoding of the number of unused bits in the last byte of the **BIT STRING**	–	07	07	–	07	07
// encoding of the authentication functional unit *(0)* The number of bits coded may vary from client to client, but within the COSEM environment, only bit 0 set to 1 (indicating the requirement of the authentication functional unit) is to be respected.	–	80	80	–	80	80
-- mechanism-name field ([11], ***IMPLICIT*** *Mechanism-name* ***OBJECT IDENTIFIER****)*						
// encoding of the tag *([11],* ***IMPLICIT****, Context-specific)*	–	8B	8B	–	8B	8B
// encoding of the length of the tagged component's value field	–	07	07	–	07	07
// encoding of the value of the **OBJECT IDENTIFIER**: - low-level-security-mechanism-name (1), - high-level-security-mechanism-name (5)	–	60 85 74 05 08 02 01	60 85 74 05 08 02 05	–	60 85 74 05 08 02 01	60 85 74 05 08 02 05

-- BER encoding of the AARQ APDU	**LN referencing**			**SN referencing**		
	no sec.	**LLS**	**HLS**	**no sec.**	**LLS**	**HLS**
*-- calling-authentication-value field ([12], Authentication-value **CHOICE**)*						
// encoding of the tag *([12], **EXPLICIT**, Context-specific)*	–	AC	AC	–	AC	AC
// encoding of the length of the tagged component's value field	–	0A	0A	–	0A	0A
// encoding of the choice for Authentication-value *(charstring [0] **IMPLICIT GraphicString**)*	–	80	80	–	80	80
// encoding of the length of the Authentication-value's value field (8 octets)	–	08	08	–	08	08
// encoding of the calling-authentication-value: - in the case of LLS, the value of the Password *"12345678"* - in the case of HLS, the value of challenge CtoS *"K56iVagY"*	–	31 32 33 34 35 36 37 38	4B 35 36 69 56 61 67 59	–	31 32 33 34 35 36 37 38	4B 35 36 69 56 61 67 59
*-- encoding of the user-information field component (Association-information, **OCTET STRING**)*						
// encoding of the tag *([30], Context-specific, Constructed)*	BE	BE	BE	BE	BE	BE
// encoding of the length of the tagged component's value field	10	10	10	10	10	10
// encoding of the choice for user-information *(**OCTET STRING**, Universal)*	04	04	04	04	04	04
// encoding of the length of the **OCTET STRING's** value field *(14 octets)*	0E	0E	0E	0E	0E	0E
// user-information: xDLMS InitiateRequest APDU	01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0			01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0		

Table D.5 – Complete AARQ APDU

LN referencing with no ciphering, lowest level security;	60 1D A1 09 06 07 60 85 74 05 08 01 01 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0
LN referencing with no ciphering, low level security;	60 36 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 31 32 33 34 35 36 37 38 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0
LN referencing with no ciphering, high level security;	60 36 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 05 AC 0A 80 08 4B 35 36 69 56 61 67 59 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0
SN referencing with no ciphering, lowest level security;	60 1D A1 09 06 07 60 85 74 05 08 01 02 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0
SN referencing with no ciphering, low level security;	60 36 A1 09 06 07 60 85 74 05 08 01 02 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 31 32 33 34 35 36 37 38 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0
SN referencing with no ciphering, high level security	60 36 A1 09 06 07 60 85 74 05 08 01 02 8A 02 07 80 8B 07 60 85 74 05 08 02 05 AC 0A 80 08 4B 35 36 69 56 61 67 59 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0

D.6 Encoding of the AARE APDU

Here, six different cases are shown:

- LN referencing with no ciphering, no security or LLS, successful establishment of the AA;
- LN referencing with no ciphering, no security or LLS, failure because the proposed application-context-name does not fit the application-context-name supported by the server (failure case 1):
  - when the meter uses LN referencing, SN referencing is proposed;
  - when the meter uses SN referencing, LN referencing is proposed;
- LN referencing with no ciphering, no security or LLS, failure because the proposed-dlms-version-number is too low; (failure case 2)
- LN referencing with no ciphering, HLS, successful establishment of the AA;
- SN referencing with no ciphering, no security or LLS, successful establishment of the AA;
- SN referencing with no ciphering, HLS, successful establishment of the AA.

The encoding is Shown in Table D.6. See also Table D.7.

Table D.6 – BER encoding of the AARE APDU

-- BER encoding of the AARE APDU	**LN referencing**				**SN referencing**	
	No sec./LLS success	**No sec./LLS failure 1**	**No sec./LLS failure 2**	**HLS success**	**No sec./LLS success**	**HLS success**
// encoding of the tag for the AARE APDU *([APPLICATION 1], Application)*	61				61	
// encoding of the length of the AARE's contents field	29	29	1F	42	29	42
-- protocol-version field ([0], ***IMPLICIT BIT STRING*** *{ version1 (0) }* ***DEFAULT*** *{ version1 }*						
// no encoding, thus it is considered with its **DEFAULT** value						
-- application-context-name field ([1], Application-context-name, ***OBJECT IDENTIFIER****)*						
// encoding of the tag *([1], Context-specific)*	A1				A1	
// encoding of the length of the tagged component's value field	09				09	
// encoding of the choice for application-context-name ***(OBJECT IDENTIFIER****, Universal)*	06				06	
// encoding of the length of the Object Identifier's value field	07				07	
// encoding of the value of the Object Identifier: NOTE when the proposed application-context does not fit the application-context supported by the server, the server responds either with the application-context name proposed or the application-context-name supported.	60 85 74 05 08 01 01	60 85 74 05 08 01 01	60 85 74 05 08 01 01 or 60 85 74 05 08 01 02	60 85 74 05 08 01 01	60 85 74 05 08 01 02	60 85 74 05 08 01 02
-- result field ([2], Association-result, ***INTEGER****)*						
// encoding of the tag *([2], Context-specific)*	A2				A2	
// encoding of the length of the tagged component's value field	03				03	
// encoding of the choice for the result *(****INTEGER****, Universal)*	02				02	
// encoding of the length of the result's value field	01				01	

-- BER encoding of the AARE APDU	**LN referencing**				**SN referencing**	
	No sec./LLS success	**No sec./LLS failure 1**	**No sec./LLS failure 2**	**HLS success**	**No sec./LLS success**	**HLS success**

// encoding of the value of the Result:
// success: 0, accepted
// failure case 1 and 2: 1, rejected-permanent
	00	01	01	00	00	00

-- result-source-diagnostic field ([3], Associate-source-diagnostic, ***CHOICE****)*						
// encoding of the tag *([3], Context-specific)*	A3				A3	
// encoding of the length of the tagged component's value field	05				05	
// encoding of the tag for the acse-service-user CHOICE *(1)*	A1				A1	
// encoding of the length of the tagged component's value field	03				03	
// encoding of the choice for associate-source-diagnostics *(INTEGER, Universal)*	02				02	
// encoding of the length of the value field	01				01	
// encoding of the value: - success, no security and LLS: 0, no diagnostics provided; - failure 1: 2, application-context-name not supported; - failure 2: 1, no-reason-given. - success, HLS security (5): 14, authentication required;	00	02	01	0E	00	0E
-- encoding of the fields of the authentication functional unit						
-- responder-acse-requirements field ([8], ***IMPLICIT,*** *ACSE-requirements,* ***BIT STRING*** *{ authentication (0) })*						
// encoding of the tag of the acse-requirements field *([8],* ***IMPLICIT,*** *Context-specific)*	–			88	–	88
// encoding of the length of the tagged component's value field.	–			02	–	02

-- BER encoding of the AARE APDU	**LN referencing**				**SN referencing**	
	No sec./LLS success	**No sec./LLS failure 1**	**No sec./LLS failure 2**	**HLS success**	**No sec./LLS success**	**HLS success**
// encoding of the number of unused bits in the last byte of the **BIT STRING**	-			07	-	07
// encoding of the authentication functional unit *(0)*	-			80	-	80
-- mechanism-name field ([9], ***IMPLICIT,*** *Mechanism-name* ***OBJECT IDENTIFIER****)*						
// encoding of the tag *([9],* ***IMPLICIT,*** *Context-specific)*	-			89	-	89
// encoding of the length of the tagged component's value field	-			07	-	07
// encoding of the value of the object identifier: high-level-security-mechanism-name (5)	-			60 85 74 05 08 02 05	-	60 85 74 05 08 02 05
-- responding-authentication-value field ([10], ***EXPLICIT,*** *Authentication-value CHOICE)*						
// encoding of the tag *([10], Context-specific)*	-	-	-	AA	-	AA
// encoding of the length of the tagged component's value field	-	-	-	0A	-	0A
// encoding of the choice for Authentication-value *(charstring [0]* ***IMPLICIT*** *GraphicString)*	-	-	-	80	-	80
// encoding of the length of the Authentication-information's value *field (8 octets)*	-	-	-	08	-	08
// encoding of the value of the challenge StoC "P6wRJ21F"	-	-	-	50 36 77 52 4A 32 31 46	-	50 36 77 52 4A 32 31 46
-- encoding of the user-information field component (Association-information, ***OCTET STRING****)*						
// encoding of the tag for the user-information field component *([30], Context-specific, Constructed)*	BE	BE	BE	BE	BE	BE
// encoding of the length of the tagged component's value field	10	10	06	10	10	10
// encoding of the choice for user-information *(OCTET STRING, Universal)*	04	04	04	04	04	04
// encoding of the length of the OCTET STRING's value field	0E	0E	04	0E	0E	0E

-- BER encoding of the AARE APDU	**LN referencing**				**SN referencing**	
	No sec./LLS success	**No sec./LLS failure 1**	**No sec./LLS failure 2**	**HLS success**	**No sec./LLS success**	**HLS success**
// failure case 1: xDLMS-InitiateResponse; // failure case 2: ConfirmedServiceError ([14]), InitiateError [1], ServiceError, initiate [6], dlms-version-too-low (1))	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	0E 01 06 01	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00	08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00

Table D.7 – The complete AARE APDU

LN referencing with no ciphering, no security or LLS, successful establishment of the AA	61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07
LN referencing with no ciphering, no security or LLS, failure because the proposed application-context-name does not fit the application-context-name supported by the server (failure case 1):	61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 01 A3 05 A1 03 02 01 02 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 or 61 29 A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 01 A3 05 A1 03 02 01 02 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07
LN referencing with no ciphering, no security or LLS, failure because the proposed-dlms-version-number is too low; (failure case 2)	61 1F A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 01 A3 05 A1 03 02 01 01 BE 06 04 04 0E 01 06 01
LN referencing with no ciphering, high level security;	61 42 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 0E 88 02 07 80 89 07 60 85 74 05 08 02 05 AA 0A 80 08 50 36 77 52 4A 32 31 46 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07
SN referencing with no ciphering, lowest level security;	61 29 A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00
SN referencing with no ciphering, high level security	61 42 A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 00 A3 05 A1 03 02 01 0E 88 02 07 80 89 07 60 85 74 05 08 02 05 AA 0A 80 08 50 36 77 52 4A 32 31 46 BE 10 04 0E 08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00

Annex E
(informative)

Encoding examples: AARQ and AARE APDUs using a ciphered application context

E.1 A-XDR encoding of the xDLMS InitiateRequest APDU, carrying a dedicated key

NOTE The System Title is the same in each example. In reality, the System Title in the request and in the response APDUs are different, as they are originated by different systems.

In this example:

- the value of the dedicated key is 00112233445566778899AABBCCDDEEFF;
- the value of the Conformance block is 007E1F;
- the value of the client-max-receive-pdu-size is 1 200 bytes (0x04B0).

The A-XDR encoding of the xDLMS InitiateRequest APDU carrying a dedicated key is shown in Table E.1.

Table E.1 – A-XDR encoding of the xDLMS InitiateRequest APDU

// encoding of the tag of the DLMS APDU CHOICE *(InitiateRequest)*	01
*-- encoding of the dedicated-key component (**OCTET STRING OPTIONAL**)*	
// usage flag *(**TRUE,** present)*	01
// length of the **OCTET STRING**	10
// contents of the **OCTET STRING**	0011223344556677 8899AABBCCDDEEFF
*-- encoding of the response-allowed component (**BOOLEAN DEFAULT TRUE**)*	
// usage flag *(**FALSE,** default value **TRUE** conveyed)*	00
-- encoding of the proposed-quality-of-service component *([0]* ***IMPLICIT*** *Integer8* ***OPTIONAL****)*	
// usage flag *(**FALSE,** not present)*	00
-- encoding of the proposed-dlms-version-number component (Unsigned8)	
// value = 6; the A-XDR encoding of an Unsigned8 is its value	06
-- encoding of the proposed-conformance component (Conformance, [APPLICATION 31] ***IMPLICIT BIT STRING*** *(SIZE(24))* [1]	
// encoding of the [APPLICATION 31] tag *(ASN.1 explicit tag)* [2]	5F1F
// encoding of the length of the 'contents' field in octet *(4)*	04
// encoding of the number of unused bits in the final octet of the BIT STRING *(0)*	00
// encoding of the fixed length BIT STRING value	007E1F
-- encoding of the client-max-receive-pdu-size component (Unsigned16)	
// value = 0x04B0, the encoding of an Unsigned16 is its value	04B0
-- resulting octet-string	0101100011223344 5566778899AABBCC DDEEFF0000065F1F 0400007E1F04B0

[1] As specified in IEC 61334-6:2000, Annex C, Examples 1 and 2, the proposed-conformance element of the xDLMS InitiateRequest APDU and the negotiated-conformance element of the xDLMS InitiateResponse APDU are encoded in BER. That's why the length of the bit-string and the number of the unused bits are encoded.

[2] For encoding of identifier octets see ISO/IEC 8825-1:2008, 8.1.2. For compliance with existing implementations, encoding of the [Application 31] tag on one byte (5F) instead of two bytes (5F 1F) is accepted when the 3-layer, connection-oriented, HDLC-based profile is used.

E.2 Authenticated encryption of the xDLMS InitiateRequest APDU

Table E.2 shows the encoding of an xDLMS InitiateRequest APDU which is also authenticated and encrypted.

Table E.2 – Authenticated encryption of the xDLMS InitiateRequest APDU

	X	Contents	LEN(X) bytes	len(X) bits
Security material				
Security suite		GCM-AES-128		
System Title	*Sys-T*	**4D4D4D**0000BC614E (here, the five last octets contain the manufacturing number in hexadecimal form)	8	64
Invocation Counter	*IC*	01234567	4	32
Initialization Vector	*IV*	*Sys-T* ǁ *IC* 4D4D4D0000BC614E01234567	12	96
Block cipher key (global)	*EK*	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	*AK*	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	16	128
Security applied		**Authenticated encryption**		
Security control byte (with unicast key)	*SC*	*SC-AE*		
		30	1	8
Security header	*SH*	*SH* = *SC-AE* ǁ *IC* 3001234567	5	40
Inputs				
xDLMS APDU to be protected	*APDU*	01011000112233445566778899AABBCC DDEEFF0000065F1F0400007E1F04B0	31	188
Plaintext	*P*	01011000112233445566778899AABBCC DDEEFF0000065F1F0400007E1F04B0	31	188
Associated data	*A*	*SC* ǁ *AK* 30D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	17	136
Outputs				
Ciphertext	*C*	801302FF8A7874133D414CED25B42534 D28DB0047720606B175BD52211BE68	31	188
Authentication tag	*T*	41DB204D39EE6FDB8E356855	12	96
The complete ciphered APDU		*TAG* ǁ *LEN* ǁ *SH* ǁ *C* ǁ *T* 2130300123456780 1302FF8A7874133D 414CED25B42534D28DB0047720606B17 5BD52211BE6841DB204D39EE6FDB8E35 6855	50	400

E.3 The AARQ APDU

In this example, the following values are used:

- Application-Context-Name: Logical_Name_Referencing_With_Ciphering;
- Calling-AP-Title (carries the System title): 4D4D4D0000BC614E;
- Mechanism-Name: COSEM_Low_Level_Security;
- Calling-Authentication-Value: 12345678.

The BER encoding of the AARQ APDU is shown in Table E.3.

Table E.3 – BER encoding of the AARQ APDU

// encoding of the tag of the AARQ APDU ([APPLICATION 0], Application)	60
// encoding of the length of the AARQ's contents field (102 octets)	66
*-- protocol-version field ([0], **IMPLICIT BIT STRING** { version 1 (0) } **DEFAULT** { version1 }*	
// no encoding, thus it is considered with its **DEFAULT** value	
-- encoding of the fields of the Kernel	
*-- application-context-name field ([1], Application-context-name, **OBJECT IDENTIFIER**)*	
// encoding of the tag ([1], Context-specific)	A1
// encoding of the length of the tagged component's value field	09
// encoding of the choice for application-context-name (**OBJECT IDENTIFIER**, Universal)	06
// encoding of the length of the Object Identifier's value field	07
// encoding of the value of the Object Identifier	60857405080103
-- encoding of the calling-AP-title field	
// encoding of the tag ([6], Context-specific)	A6
// encoding of the length of the tagged component's value field	0A
// encoding of the type ([4], Universal, Octetstring type)	04
// encoding of the length of the calling-AP-title-field	08
// encoding of the value	4D4D4D0000BC614E
-- encoding of the fields of the authentication functional unit	
--sender-acse-requirements field ([10], ACSE-requirements, **BIT STRING** { authentication (0) })	
// encoding of the tag of the acse-requirements field ([10], **IMPLICIT**, Context-specific)	8A
// encoding of the length of the tagged component's value field	02
// encoding of the number of unused bits in the last byte of the **BIT STRING**	07
// encoding of the authentication functional unit (0) The number of bits coded may vary from client to client, but within the COSEM environment, only bit 0 set to 1 (indicating the requirement of the authentication functional unit) is to be respected.	80
*-- mechanism-name field ([11], **IMPLICIT** Mechanism-name **OBJECT IDENTIFIER**)*	
// encoding of the tag ([11], **IMPLICIT**, Context-specific)	8B
// encoding of the length of the tagged component's value field	07
// encoding of the value of the **OBJECT IDENTIFIER:** - low-level-security-mechanism-name,	60857405080201
*-- calling-authentication-value field ([12], Authentication-value **CHOICE**)*	
// encoding of the tag ([12], **EXPLICIT**, Context-specific)	AC
// encoding of the length of the tagged component's value field	0A
// encoding of the choice for Authentication-value (charstring [0] **IMPLICIT GraphicString**)	80
// encoding of the length of the Authentication-value's value field (8 octets)	08
// encoding of the calling-authentication-value (12345678)	3132333435363738
*-- encoding of the user-information field component (Association-information, **OCTET STRING**)*	

// encoding of the tag ([30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	34
// encoding of the choice for user-information (**OCTET STRING**, Universal)	04
// encoding of the length of the **OCTET STRING's** value field (32 octets)	32
-- ciphered xDLMS InitiateRequest APDU	2130300123456780 1302FF8A7874133D 414CED25B42534D2 8DB0047720606B17 5BD52211BE6841DB 204D39EE6FDB8E35 6855

E.4 A-XDR encoding of the xDLMS InitiateResponse APDU

In this example:

- the value of the Conformance block is 007C1F;
- the value of the server-max-receive-pdu-size is 1 024 bytes (0x0400).

The A-XDR encoding of the xDLMS InitiateResponse APDU is shown in Table E.4.

Table E.4 – A-XDR encoding of the xDLMS InitiateResponse APDU

// encoding of the tag of the DLMS APDU CHOICE (InitiateResponse)	08
*-- encoding of the negotiated-quality-of-service component ([0] **IMPLICIT** Integer8 **OPTIONAL**)*	
// usage flag (**FALSE**, not present)	00
-- encoding of the negotiated-dlms-version-number component (Unsigned8)	
// value = 6, the A-XDR encoding of an Unsigned8 is its value	06
*-- encoding of the negotiated-conformance component (Conformance, [APPLICATION 31] **IMPLICIT BIT STRING** (SIZE(24))*	
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag)	5F1F
// encoding of the length of the 'contents' field in octet (4)	04
// encoding of the number of unused bits in the final octet of the BIT STRING (0)	00
// encoding of the fixed length BIT STRING value	007C1F
-- encoding of the server-max-receive-pdu-size component (Unsigned16)	
// value = 0x0400, the encoding of an Unsigned16 is its value	0400
-- encoding of the VAA-Name component (ObjectName, Integer16)	
// value=0x0007; the encoding of a value constrained Integer16 is its value	0007
-- resulting octet-string, to be inserted in the user-information field of the AARE APDU	0800065F1F040000 7C1F04000007

E.5 Authenticated encryption of the xDLMS InitiateResponse APDU

Table E.5 shows the encoding of the xDLMS InitiateResponse APDU which is also authenticated and encrypted.

Table E.5 – Authenticated encryption of the xDLMS InitiateResponse APDU

	X	Contents	LEN(X) bytes	len(X) bits
Security material				
Security suite		GCM-AES-128		
System Title	*Sys-T*	4D4D4D0000BC614E (here, the five last octets contain the manufacturing number in hexadecimal form)	8	64
Invocation Counter	*IC*	01234567	4	32
Initialization Vector	*IV*	*Sys-T* II *IC*		
		4D4D4D0000BC614E01234567	12	96
Block cipher key (global)	*EK*	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	*AK*	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	16	128
Security applied		**Authenticated encryption**		
Security control byte (with unicast key)	*SC*	*SC-AE*		
		30	1	8
Security header	*SH*	*SH* = *SC-AE* II *IC*		
		3001234567	5	40
Inputs				
xDLMS APDU to be protected	*APDU*	0800065F1F0400007C1F04000007	14	112
Plaintext	*P*	0800065F1F0400007C1F04000007	14	112
Associated data	*A*	*SC* II *AK* 30D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	17	136
Outputs				
Ciphertext	*C*	891214A0845E475714383F65BC19	14	112
Authentication tag	*T*	745CA235906525E4F3E1C893	12	96
The complete Ciphered APDU		*TAG* II *LEN* II *SH* II *C* II *T* 281F3001234567891214A0845E475714 383F65BC19745CA235906525E4F3E1C8 93	33	264

E.6 The AARE APDU

The BER encoding of the AARE APDU is shown in Table E.6.

Table E.6 – BER encoding of the AARE APDU

-- BER encoding of the AARE APDU	
// encoding of the tag for the AARE APDU ([APPLICATION 1], Application)	61
// encoding of the length of the AARE's contents field (72 octets)	48
-- protocol-version field ([0], ***IMPLICIT BIT STRING*** *{ version1 (0) }* ***DEFAULT*** *{ version1 }*	
// no encoding, thus it is considered with its **DEFAULT** value	
-- encoding of the fields of the Kernel	
-- application-context-name field ([1], Application-context-name, **OBJECT IDENTIFIER**)	
// encoding of the tag ([1], Context-specific)	A1
// encoding of the length of the tagged component's value field	09
// encoding of the choice for application-context-name **(OBJECT IDENTIFIER**, Universal)	06
// encoding of the length of the Object Identifier's value field	07
// encoding of the value of the Object Identifier: NOTE When the proposed application-context does not fit the application-context supported by the server, the server responds either with the application-context name proposed or the application-context-name supported.	60857405080103
-- result field ([2], Association-result, ***INTEGER****)*	
// encoding of the tag ([2], Context-specific)	A2
// encoding of the length of the tagged component's value field	03
// encoding of the choice for the result (**INTEGER**, Universal)	02
// encoding of the length of the result's value field	01
// encoding of the value of the Result: 0, accepted	00
-- result-source-diagnostic field ([3], Associate-source-diagnostic, ***CHOICE****)*	
// encoding of the tag ([3], Context-specific)	A3
// encoding of the length of the tagged component's value field	05
// encoding of the tag for the acse-service-user CHOICE (1)	A1
// encoding of the length of the tagged component's value field	03
// encoding of the choice for associate-source-diagnostics (INTEGER, Universal)	02
// encoding of the length of the value field	01
// encoding of the value (0, no diagnostics provided)	00
-- encoding of the responding-AP-title field	
// encoding of the tag ([4], Context-specific)	A4
// encoding of the length of the tagged component's value field	0A
// encoding of the type ([4], Universal, Octetstring type)	04
// encoding of the length of the responding-AP-title-field	08
// encoding of the value	4D4D4D0000BC614E
-- encoding of the fields of the authentication functional unit -- In this example the Authentication functional unit is not -- present; it is not necessary in the case of LLS, but if it is -- present it is also acceptable.	
-- encoding of the user-information field component (Association-information, ***OCTET STRING****)*	
// encoding of the tag ([30], Context-specific, Constructed)	BE

-- BER encoding of the AARE APDU	
// encoding of the length of the tagged component's value field	23
// encoding of the choice for user-information (**OCTET STRING**, Universal)	04
// encoding of the length of the **OCTET STRING's** value field	21
-- *ciphered xDLMS InitiateResponse APDU*	281F300123456789 1214A0845E475714 383F65BC19745CA2 35906525E4F3E1C8 93

E.7 The RLRQ APDU (carrying a ciphered xDLMS InitiateRequest APDU)

The BER encoding of the RLRQ APDU is shown in Table E.7.

Table E.7 – BER encoding of the RLRQ APDU

-- BER encoding of the RLRQ APDU	
// encoding of the tag of the RLRQ APDU ([APPLICATION 2], Application)	62
// encoding of the length of the RLRQ's contents field	39
-- *reason field*	
// encoding of the tag ([0], IMPLICIT]	80
// encoding of the length of the tagged component's value field	01
// encoding of the value (0, normal)	00
-- *encoding of the user-information field component (Association-information,* ***OCTET STRING****)*	
// encoding of the tag ([30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	34
// encoding of the choice for user-information (**OCTET STRING**, Universal)	04
// encoding of the length of the **OCTET STRING's** value field (14 octets)	32
// user-information: xDLMS InitiateRequest APDU	2130300123456780 1302FF8A7874133D 414CED25B42534D2 8DB0047720606B17 5BD52211BE6841DB 204D39EE6FDB8E35 6855

E.8 The RLRE APDU (carrying a ciphered xDLMS InitiateResponse APDU)

The BER encoding of the RLRQ APDU is shown in Table E.8.

Table E.8 – BER encoding of the RLRE APDU

-- BER encoding of the RLRE APDU	
// encoding of the tag of the RLRE APDU ([APPLICATION 3], Application)	63
// encoding of the length of the RLRE's contents field	28
-- reason field	
// encoding of the tag ([0], IMPLICIT]	80
// encoding of the length of the tagged component's value field	01
// encoding of the value (0, normal)	00
*-- encoding of the user-information field component (Association-information, **OCTET STRING**)*	
// encoding of the tag ([30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	23
// encoding of the choice for user-information (**OCTET STRING**, Universal)	04
// encoding of the length of the **OCTET STRING's** value field (14 octets)	21
// user-information: xDLMS InitiateResponse APDU	281F300123456789 1214A0845E475714 383F65BC19745CA2 35906525E4F3E1C8 93

Annex F
(informative)

Data transfer service examples

F.1 GET / Read, SET / Write examples

Table F.2 to Table F.9 show examples for data exchange using xDLMS services with LN referencing (left column) and SN referencing (right column). Table F.1 shows the objects used in the examples.

Table F.1 – The objects used in the examples

```
Object 1:

-   Class: Data

-   Logical name: 0000800000FF

-   Short name of value attribute: 0100
-   Value: octet string of 50 elements

-   0102030405060708091011121314151 6

-   17181920212223242526272829303132

-   33343536373839404142434445464748

-   4950


Object 2:

-   Class: Data

-   Logical name: 0000800100FF

-   Short name of value attribute: 0110

-   Value: visible string of 3 elements 303030
```

In the case of block transfer, the negotiated APDU size is 40 bytes.

NOTE What is negotiated is the APDU size, not the block size. Therefore, the block size is smaller than the APDU size.

Table F.2 – Example: Reading the value of a single attribute without block transfer

C001C1 00010000800000FF0200 <GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> </GetRequest>	0501 020100 <ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest>

C401C1
00
0932
0102030405060708091011121314151 6
17181920212223242526272829303132
33343536373839404142434445464748
4950
 	
<GetResponse>
  <GetResponsenormal>
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <Data>
        <OctetString Value="0102030405060708091011121314151 6
                            17181920212223242526272829303132
                            33343536373839404142434445464748
                            4950" />
      </Data>
    </Result>
  </GetResponsenormal>
	0C01
	00
	0932
	0102030405060708091011121314151 6
	17181920212223242526272829303132
	33343536373839404142434445464748
	4950
	
	<ReadResponse Qty="0001" >
	  <Data>
	    <OctetString Value="0102030405060708091011121314151 6
	                        17181920212223242526272829303132
	                        33343536373839404142434445464748
	                        4950" />
	  </Data>
	</ReadResponse>

</GetResponse>	

Table F.3 – Example: Reading the value of a list of attributes without block transfer

```
C003C1

02

00010000800000FF0200

00010000800100FF0200


  <GetRequestWithList>

    <InvokeIdAndPriority Value="C1" />

    <AttributeDescriptorList Qty="0002" >

      <_AttributeDescriptorWithSelection>

        <AttributeDescriptor>

          <ClassId Value="0001" />

          <InstanceId Value="0000800000FF" />

          <AttributeId Value="02" />

        </AttributeDescriptor>

      </_AttributeDescriptorWithSelection>

      <_AttributeDescriptorWithSelection>

        <AttributeDescriptor>

          <ClassId Value="0001" />

          <InstanceId Value="0000800100FF" />

          <AttributeId Value="02" />
```

```
0502

020100

020110


<ReadRequest Qty="0002" >

  <VariableName Value="0100" />

  <VariableName Value="0110" />

</ReadRequest>
```

</AttributeDescriptor> </_AttributeDescriptorWithSelection> </AttributeDescriptorList> </GetRequestWithList> </GetRequest>	

```
C403C1

02

00

0932

0102030405060708091011121314151 6

171819202122232425262728293031 32

333435363738394041424344454647 48

4950

00

0A03

303030


<GetResponse>

  <GetResponseWithList>

    <InvokeIdAndPriority Value="C1" />

    <Result Qty="0002" >

      <Data>

        <OctetString Value="01020304050607080910111213141516

                            17181920212223242526272829303132

                            33343536373839404142434445464748
```

```
0C02

00

0932

01020304050607080910111213141516

17181920212223242526272829303132

33343536373839404142434445464748

4950

00

0A03

303030


<ReadResponse Qty="0002" >

  <Data>

    <OctetString Value="01020304050607080910111213141516

                        17181920212223242526272829303132

                        33343536373839404142434445464748

                        4950" />

  </Data>

  <Data>

    <VisibleString Value="303030" />
```

4950" /> </Data> <Data> <VisibleString Value="303030" /> </Data> </Result> </GetResponseWithList> </GetResponse>	</Data> </ReadResponse>

Table F.4 – Example: Reading the value of a single attribute with block transfer

C001C1	0501
00010000800000FF0200	020100
<GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> </GetRequest>	<ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest>

```
C402C1
00
00000001
00
1E
09320102030405060708091011121314
15161718192021222324252627 28

<GetResponse>
  <GetResponsewithDataBlock>
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <LastBlock Value="00" />
      <BlockNumber Value="00000001" />
      <Result>
        <RawData Value="0932010203040506070809101112131 4
                        15161718192021222324252627 28" />
      </Result>
    </Result>
  </GetResponsewithDataBlock>
```

```
0C01
02
00
0001
21
01000932010203040506070809101112
13141516171819202122232425262728
29

<ReadResponse Qty="0001" >
  <DataBlockResult>
    <LastBlock Value="00" />
    <BlockNumber Value="0001" />
    <RawData Value="01000932010203040506070809101112
                    13141516171819202122232425262728
                    29" />
  </DataBlockResult>
</ReadResponse>

// 33 bytes of raw-data contains number of data, success, data
```

</GetResponse>
 	
// 30 bytes of raw-data contains data type, length and 28 bytes
 	
// of data. Note that Data is encoded, not Get-Data-result.
	// type, length and 29 bytes of data.
	
	// As the raw-data contains the data encoded exactly as without
	
	// block transfer, the number of results is encoded because the
	
	// ReadResponse is a **SEQUENCE OF CHOICE**.

C002C1 00000001 <GetRequest> <GetRequestForNextDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </GetRequestForNextDataBlock> </GetRequest>	0501 050001 <ReadRequest Qty="0001" > <BlockNumberAccess> <BlockNumber Value="0001" /> </BlockNumberAccess> </ReadRequest>

```
C402C1

01

00000002

00

16

2930313233343536373839404142434 4

454647484950


<GetResponse>

  <GetResponsewithDataBlock>

    <InvokeIdAndPriority Value="C1" />

    <Result>

      <LastBlock Value="01" />

      <BlockNumber Value="00000002" />

      <Result>

        <RawData Value="29303132333435363738394041424344

                        454647484950" />

      </Result>

    </Result>

  </GetResponsewithDataBlock>
```

```
0C01

02

01

0002

15

303132333435363738394041424344 45

4647484950


<ReadResponse Qty="0001" >

  <DataBlockResult>

    <LastBlock Value="01" />

    <BlockNumber Value="0002" />

    <RawData Value="303132333435363738394041424344 45

                    4647484950" />

  </DataBlockResult>

</ReadResponse>


// APDU length 28 bytes, 21 bytes of raw-data carries the

// remaining part of data requested.
```

</GetResponse>
 	
// APDU length 32 bytes, 22 bytes of raw-data carries the
// remaining part of data requested.	

Table F.5 – Example: Reading the value of a list of attributes with block transfer

C003C1 02 00010000800000FF0200 00010000800100FF0200 <GetRequestWithList> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" > <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800100FF" /> <AttributeId Value="02" />	0502 020100 020110 <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest>

</AttributeDescriptor> </_AttributeDescriptorWithSelection> </AttributeDescriptorList> </GetRequestWithList> </GetRequest>	

C402C1	0C01
00	02
00000001	00
00	0001
1E	21
02000932010203040506070809101112	02000932010203040506070809101112
1314151617181920212223242526	13141516171819202122232425262728
	29
<GetResponse>	
<GetResponsewithDataBlock>	<ReadResponse Qty="0001" >
<InvokeIdAndPriority Value="C1" />	<DataBlockResult>
<Result>	<LastBlock Value="00" />
<LastBlock Value="00" />	<BlockNumber Value="0001" />
<BlockNumber Value="00000001" />	<RawData Value="02000932010203040506070809101112
<Result>	13141516171819202122232425262728
<RawData Value="02000932010203040506070809101112	29" />
1314151617181920212223242526" />	</DataBlockResult>
</Result>	</ReadResponse>
</Result>	
</GetResponsewithDataBlock>	// 33 bytes of raw-data contains the number of results and part

</GetResponse>
 	
// 30 bytes of raw-data contains the number of results and part
// of the data. The first one is success, octet-string of 32
// elements; the first 26 bytes fit in.
	// of the data. The first one is success, octet-string of 32
	// elements; the first 29 bytes fit in.

C002C1 00000001 <GetRequest> <GetRequestForNextDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </GetRequestForNextDataBlock> </GetRequest>	0501 05 0001 <ReadRequest Qty="0001" > <BlockNumberAccess> <BlockNumber Value="0001" /> </BlockNumberAccess> </ReadRequest>

```
C402C1

01

00000002

00

1E

27282930313233343536373839404142

4344454647484950000A03303030


<GetResponse>

  <GetResponsewithDataBlock>

    <InvokeIdAndPriority Value="C1" />

    <Result>

      <LastBlock Value="01" />

      <BlockNumber Value="00000002" />

      <Result>

        <RawData Value="27282930313233343536373839404142

                        4344454647484950000A03303030" />

      </Result>

    </Result>

  </GetResponsewithDataBlock>
```

```
0C01

02

01

0002

1B

30313233343536373839404142434445

4647484950000A03303030


<ReadResponse Qty="0001" >

  <DataBlockResult>

    <LastBlock Value="01" />

    <BlockNumber Value="0002" />

    <RawData Value="30313233343536373839404142434445

                    4647484950000A03303030" />

  </DataBlockResult>

</ReadResponse>


// The APDU is 34 bytes. It contains the second and last block.

// 27 bytes of raw-data contains the remaining 21 bytes of the
```

</GetResponse>
 	
// 30 bytes of raw-data contains the remaining 24 bytes of the
 	
// first result and it also contains the six bytes of the second
 	
//result: success, visible string of 3 elements.
	// first result and the six bytes of the second result: success, // visible string of 3 elements

Table F.6 – Example: Writing the value of a single attribute without block transfer

```
C101C1
00010000800000FF0200
0932
0102030405060708091011121314151
6
```

```
C101C1
00010000800000FF0200
0932
01020304050607080910111213141516
17181920212223242526272829303132
33343536373839404142434445464748
4950

<SetRequest>
  <SetRequestNormal>
    <InvokeIdAndPriority Value="C1" />
    <AttributeDescriptor>
      <ClassId Value="0001" />
      <InstanceId Value="0000800000FF" />
      <AttributeId Value="02" />
    </AttributeDescriptor>
    <Value>
      <OctetString Value="01020304050607080910111213141516
                          17181920212223242526272829303132
                          33343536373839404142434445464748
                          4950" />
    </Value>
  </SetRequestNormal>
</SetRequest>
```

```
06
01
020100
01
0932
01020304050607080910111213141516
17181920212223242526272829303132
33343536373839404142434445464748
4950

<WriteRequest>
  <ListOfVariableAccessSpecification Qty="0001" >
    <VariableName Value="0100" />
  </ListOfVariableAccessSpecification>
  <ListOfData Qty="0001" >
    <OctetString Value="01020304050607080910111213141516
                        17181920212223242526272829303132
                        33343536373839404142434445464748
                        4950" />
  </ListOfData>
</WriteRequest>
```

C501C1 00 <SetResponse> <SetResponseNormal> <InvokeIdAndPriority Value="C1" /> <Result Value="Success" /> </SetResponseNormal> </SetResponse>	0D01 00 <WriteResponse Qty="0001" > <Success /> </WriteResponse>

Table F.7 – Example: Writing the value of a list of attributes without block transfer

```
C104C1

02

00010000800000FF0200

00010000800100FF0200

02

0932

0102030405060708091011121314151 6

17181920212223242526272829303132

33343536373839404142434445464748

4950

0A03

303030


<SetRequest>

  <SetRequestNormalWithList>

    <InvokeIdAndPriority Value="C1" />

    <AttributeDescriptorList Qty="0002" >

      <_AttributeDescriptorWithSelection>

        <AttributeDescriptor>

          <ClassId Value="0001" />
```

```
0602

020100

020110

02

0932

01020304050607080910111213141516

17181920212223242526272829303132

33343536373839404142434445464748

4950

0A03

303030


<WriteRequest>

  <ListOfVariableAccessSpecification Qty="0002" >

    <VariableName Value="0100" />

    <VariableName Value="0110" />

  </ListOfVariableAccessSpecification>

  <ListOfData Qty="0002" >

    <OctetString Value="01020304050607080910111213141516

                        17181920212223242526272829303132
```

```
        <InstanceId Value="0000800000FF" />
        <AttributeId Value="02" />
      </AttributeDescriptor>
    </_AttributeDescriptorWithSelection>
    <_AttributeDescriptorWithSelection>
      <AttributeDescriptor>
        <ClassId Value="0001" />
        <InstanceId Value="0000800100FF" />
        <AttributeId Value="02" />
      </AttributeDescriptor>
    </_AttributeDescriptorWithSelection>
  </AttributeDescriptorList>
  <ValueList Qty="0002" >
    <OctetString Value="01020304050607080910111213141516
                        17181920212223242526272829303132
                        33343536373839404142434445464748
                        4950" />
    <VisibleString Value="303030" />
  </ValueList>
</SetRequestNormalWithList>
```

```
                        33343536373839404142434445464748
                        4950" />
    <VisibleString Value="303030" />
  </ListOfData>
</WriteRequest>
```

</SetRequest>	
C505C1 02 00 00 <SetResponse> <SetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <_DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> </SetResponseWithList> </SetResponse>	0D02 00 00 <WriteResponse Qty="0002" > <Success /> <Success /> </WriteResponse>

Table F.8 – Example: Writing the value of a single attribute with block transfer

C102C1 00010000800000FF0200 00 00000001 15 09320102030405060708091011121314 1516171819 <SetRequest> <SetRequestWithFirstDataBlock> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> <DataBlock> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <RawData Value="09320102030405060708091011121314	0601 07 00 0001 01 091F 0102010001093201020304050607080 1011121314151617181920212223 24 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> <BlockNumber Value="0001" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="01020100010932010203040506070809 101112131415161718192021222324" /> </ListOfData>

1516171819" />
</DataBlock>
</SetRequestWithFirstDataBlock>
</SetRequest>
 	
// 21 bytes of raw-data contain the type, length and the first 19
// bytes of data to be written.
	</WriteRequest>
	
	// 31 bytes of octet-string contains raw-data: the sequence of
	// Variable-Access-Specification, the sequence of data, the type,
	// length and the first 24 bytes to be written.

C502C1 00000001 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0001 <WriteResponse Qty="0001" > <BlockNumber Value="0001" /> </WriteResponse>

```
C103C1

01

00000002

1F

20212223242526272829303132333435

363738394041424344454647484950



<SetRequest>

  <SetRequestWithDataBlock>

    <InvokeIdAndPriority Value="C1" />

    <DataBlock>

      <LastBlock Value="01" />

      <BlockNumber Value="00000002" />

      <RawData Value="20212223242526272829303132333435

                      363738394041424344454647484950" />

    </DataBlock>

  </SetRequestWithDataBlock>

</SetRequest>



// 31 bytes of raw-data contains the remaining 21 bytes of the
```

```
0601
07
01
0002
01
091A
25262728293031323334353637383940
41424344454647484950


<WriteRequest>

  <ListOfVariableAccessSpecification Qty="0001" >

    <WriteDataBlockAccess>

      <LastBlock Value="01" />

      <BlockNumber Value="0002" />

    </WriteDataBlockAccess>

  </ListOfVariableAccessSpecification>
  <ListOfData Qty="0001" >

    <OctetString Value="25262728293031323334353637383940

                        41424344454647484950" />

  </ListOfData>

</WriteRequest>

// The APDU is 35 bytes. 26 bytes of octet-string contains raw-

// data: the remaining 26 bytes of data to be written.
```

// data to be written.	
C503C10000000002 <SetResponse> <SetResponseForLastDataBlock> <InvokeIdAndPriority Value="C1" /> <Result Value="Success" /> <BlockNumber Value="00000002" /> </SetResponseForLastDataBlock> </SetResponse>	0D0100 <WriteResponse Qty="0001" > <Success /> </WriteResponse>

Table F.9 – Example: Writing the value of a list of attributes with block transfer

C105C1	0601
02	07
00010000800000FF0200	00
00010000800100FF0200	0001
00	01
00000001	091F
0A	020201000201100209320102030405 06
02093201020304050607	07080910111213141516171819202 1
<SetRequest>	<WriteRequest>
<SetRequestWithListAndWithFirstDatablock>	<ListOfVariableAccessSpecification Qty="0001" >
<InvokeIdAndPriority Value="C1" />	<WriteDataBlockAccess>
<AttributeDescriptorList Qty="0002" >	<LastBlock Value="00" />
<_AttributeDescriptorWithSelection>	<BlockNumber Value="0001" />
<AttributeDescriptor>	</WriteDataBlockAccess>
<ClassId Value="0001" />	</ListOfVariableAccessSpecification>
<InstanceId Value="0000800000FF" />	<ListOfData Qty="0001" >
<AttributeId Value="02" />	<OctetString Value="02020100020110020932010203040506
</AttributeDescriptor>	0708091011121314151617181920 21" />
</_AttributeDescriptorWithSelection>	</ListOfData>

```
      <_AttributeDescriptorWithSelection>
        <AttributeDescriptor>
          <ClassId Value="0001" />
          <InstanceId Value="0000800100FF" />
          <AttributeId Value="02" />
        </AttributeDescriptor>
      </_AttributeDescriptorWithSelection>
    </AttributeDescriptorList>
    <DataBlock>
      <LastBlock Value="00" />
      <BlockNumber Value="00000001" />
      <RawData Value="02093201020304050607" />
    </DataBlock>
  </SetRequestWithListAndWithFirstDatablock>
</SetRequest>

// The APDU is 40 bytes. It contains the two attribute
// descriptors and 10 bytes of raw-data containing the type and
// length of the first data and the first 7 bytes to be written.
```

```
</WriteRequest>

// The APDU is 40 bytes. 31 bytes of octet-string contains raw-
// data: the number and the name of objects to be written, the
// number of data to be written and the first 21 bytes of the
// first data to be written.
```

C502C1 00000001 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0001 <WriteResponse Qty="0001" > <BlockNumber Value="0001" /> </WriteResponse>

```
C103C1

00

00000002

1F

08091011121314151617181920212223

242526272829303132333435363738


<SetRequest>

  <SetRequestWithDataBlock>

    <InvokeIdAndPriority Value="C1" />

    <DataBlock>

      <LastBlock Value="00" />

      <BlockNumber Value="00000002" />

      <RawData Value="08091011121314151617181920212223

                      242526272829303132333435363738" />

    </DataBlock>

  </SetRequestWithDataBlock>

</SetRequest>


// The APDU is 40 bytes. 31 bytes of raw-data contain the second
```

```
0601

07

00

0002

01

091F

22232425262728293031323334353637

383940414243444546474849500A03


<WriteRequest>

  <ListOfVariableAccessSpecification Qty="0001" >

    <WriteDataBlockAccess>

      <LastBlock Value="00" />

      <BlockNumber Value="0002" />

    </WriteDataBlockAccess>

  </ListOfVariableAccessSpecification>

  <ListOfData Qty="0001" >

    <OctetString Value="22232425262728293031323334353637

                        383940414243444546474849500A03" />

  </ListOfData>
```

// part of data to be written.	</WriteRequest>
	
	// The APDU is 40 bytes. 31 bytes of octet-string contains raw-
	// data: the second 29 bytes of the first data to be written and
	// the data type and length of the second data to be written. The
	// value follows in the next block.

C502C100000002 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000002" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0002 <WriteResponse Qty="0001" > <BlockNumber Value="0002" /> </WriteResponse>

```
C103C1
01
00000003
11
3940414243444546474849500A033030
30


<SetRequest>
  <SetRequestWithDataBlock>
    <InvokeIdAndPriority Value="C1" />
    <DataBlock>
      <LastBlock Value="01" />
      <BlockNumber Value="00000003" />
      <RawData Value="3940414243444546474849500A033030
                      30" />
    </DataBlock>
  </SetRequestWithDataBlock>
</SetRequest>
```

```
0601
07
01
0003
01
0903
303030


<WriteRequest>
  <ListOfVariableAccessSpecification Qty="0001" >
    <WriteDataBlockAccess>
      <LastBlock Value="01" />
      <BlockNumber Value="0003" />
    </WriteDataBlockAccess>
  </ListOfVariableAccessSpecification>
  <ListOfData Qty="0001" >
    <OctetString Value="303030" />
  </ListOfData>
</WriteRequest>
```

// The APDU is 26 bytes. 17 bytes of raw-data contain the third // part of the first data and the second data to be written.	// The APDU is 12 bytes. 3 bytes of octet-string contains // raw-data: the value of the second attribute.
C504C1 02 00 00 00000003 <SetResponse> <SetResponseForLastDataBlockWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <_DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> <BlockNumber Value="00000003" /> </SetResponseForLastDataBlockWithList> </SetResponse>	0D02 00 00 <WriteResponse Qty="0002" > <Success /> <Success /> </WriteResponse>

F.2 ACCESS service example

Table F.10 shows an example of the ACCESS service without general block transfer.

Table F.10 – Example: ACCESS service without block transfer

Message Elements (MAX APDU = 1024)	Contents	LEN (Bytes)
Access-Request	D9	1
long-invoke-id-and-priority	40000000	4
date-time	00	1
access-request-body		0
access-request-specification		0
SEQUENCE OF CHOICE	04	1
access-request-get	01	1
cosem-attribute-descriptor		0
class-id	0001	2
instance-id	0000600100FF	6
attr-id	02	1
access-request-get	01	1
cosem-attribute-descriptor		0
class-id	0008	2
instance-id	0000010000FF	6
attr-id	02	1
access-request-set	02	1
cosem-attribute-descriptor		0
class-id	0014	2
instance-id	00000D0000FF	6
attr-id	07	1
access-request-set	02	1
cosem-attribute-descriptor		0
class-id	0014	2
instance-id	00000D0000FF	6
attr-id	08	1
access-request-list-of-data		
SEQUENCE OF Data	04	1
null-data	00	1
null-data	00	1
array	01040203090100090CFFFFFFFFFFFFFF000 00000000901FF0203090101090CFFFFFFFF FFFFFF00000000000901FF0203090102090 CFFFFFFFFFFFFFF00000000000901FF0203 090103090CFFFFFFFFFFFFFF00000000000 901FF	90

array	0104020809010011FF11FF11FF11FF11FF1 1FF11FF02080901011102110111011101111 01110111010208090102 11FF11FF11FF11F F11FF11FF11FF0208090103110111021102 1102110211021102	78
Complete Access-Request APDU (encoded)	D940000000000040100010000600100FF020 10008000001000 0FF0202001400000D0000 FF0702001400000D0000FF0804000001040 203090100090CFFFFFFFFFFFFFF00000000 000901FF0203090101090CFFFFFFFFFFFFF F00000000000901FF0203090102090CFFFF FFFFFFFFFF00000000000901FF020309010 3090CFFFFFFFFFFFFFF00000000000901FF 0104020809010011FF11FF11FF11FF11FF1 1FF11FF02080901011102110111011101111 01110111010208090102 11FF11FF11FF11F F11FF11FF11FF0208090103110111021102 1102110211021102	218
Access-Response	DA	1
long-invoke-id-and-priority	40000000	4
date-time	00	1
access-response-body		0
access-request-specification OPTIONAL	00	1
access-response-list-of-data		0
SEQUENCE OF Data	04	1
octet-string	09083030303030303031	10
octet-string	090C07DC030C07161E0000FF8880	14
null-data	00	1
null-data	00	1
access-response-specification		0
SEQUENCE OF CHOICE	04	1
access-response-get	01	1
result	00	1
access-response-get	01	1
result	00	1
access-response-set	02	1
result	00	1
access-response-set	02	1
result	00	1
Complete Access-Response APDU (encoded)	DA40000000000004090830303030303030303 1090C07DC030C07161E0000FF8880000004 010001000200020 0	43

F.3 Compact array encoding example

F.3.1 General

Any series of data of the same type can be encoded as array or compact-array.

The compact-array data type is present from the beginning in the DLMS®/COSEM specification, but so far its interpretation was not unambiguous, therefore it has not been used.

The objective of this subclause is to facilitate the use of compact-array encoding.

F.3.2 The specification of compact-array

Subclause 7.3.13 specifies the following:

```
Data::= CHOICE
{
    null-data                       [0]   IMPLICIT   NULL,
    array                           [1]   IMPLICIT   SEQUENCE OF Data,
    structure                       [2]   IMPLICIT   SEQUENCE OF Data,
    boolean                         [3]   IMPLICIT   BOOLEAN,
    bit-string                      [4]   IMPLICIT   BIT STRING,
    double-long                     [5]   IMPLICIT   Integer32,
    double-long-unsigned            [6]   IMPLICIT   Unsigned32,
    floating-point                  [7]   IMPLICIT   OCTET STRING(SIZE(4)) ,
    octet-string                    [9]   IMPLICIT   OCTET STRING,
    visible-string                  [10]  IMPLICIT   VisibleString,
    bcd                             [13]  IMPLICIT   Integer8,
    utf8-string                     [12]  IMPLICIT   NULL,
    integer                         [15]  IMPLICIT   Integer8,
    long                            [16]  IMPLICIT   Integer16,
    unsigned                        [17]  IMPLICIT   Unsigned8,
    long-unsigned                   [18]  IMPLICIT   Unsigned16,
    compact-array                   [19]  IMPLICIT   SEQUENCE
    {
        contents-description        [0]              TypeDescription,
        array-contents              [1]   IMPLICIT   OCTET STRING
    },
    long64                          [20]  IMPLICIT   Integer64,
```

```
    long64-unsigned                     [21] IMPLICIT  Unsigned64,
    enum                                [22] IMPLICIT  Unsigned8,
    float32                             [23] IMPLICIT  OCTET STRING (SIZE(4)),
    float64                             [24] IMPLICIT  OCTET STRING (SIZE(8)),
    date_time                           [25] IMPLICIT  OCTET STRING (SIZE(12)),
    date                                [26] IMPLICIT  OCTET STRING (SIZE(5)),
    time                                [27] IMPLICIT  OCTET STRING (SIZE(4)),
    dont-care                           [255] IMPLICIT NULL
}

-- The following TypeDescription relates to the compact-array data Type

TypeDescription::= CHOICE
{
    null-data                           [0]  IMPLICIT NULL,
    array                               [1]  IMPLICIT SEQUENCE
    {
        number-of-elements      Unsigned16,
        type-description        TypeDescription
    },
    structure                           [2]  IMPLICIT SEQUENCE OF TypeDescription,
    boolean                             [3]  IMPLICIT NULL,
    bit-string                          [4]  IMPLICIT NULL,
    double-long                         [5]  IMPLICIT NULL,
    double-long-unsigned                [6]  IMPLICIT NULL,
    floating-point                      [7]  IMPLICIT NULL,
    octet-string                        [9]  IMPLICIT NULL,
    visible-string                      [10] IMPLICIT NULL,
    bcd                                 [13] IMPLICIT NULL,
    integer                             [15] IMPLICIT NULL,
    long                                [16] IMPLICIT NULL,
    unsigned                            [17] IMPLICIT NULL,
```

```
    long-unsigned                       [18]  IMPLICIT  NULL,



    long64                              [20]  IMPLICIT  NULL,

    long64-unsigned                     [21]  IMPLICIT  NULL,

    enum                                [22]  IMPLICIT  NULL,

    float32                             [23]  IMPLICIT  NULL,

    float64                             [24]  IMPLICIT  NULL,

    date_time                           [25]  IMPLICIT  NULL,

    date                                [26]  IMPLICIT  NULL,

    time                                [27]  IMPLICIT  NULL,

    dont-care                           [255] IMPLICIT  NULL
}
```

Notice that in the compact-array type:

- contents-description / TypeDescription specifies the data type of the elements in the compact array:
- in the case of simple data types it contains the tag of the type. In the case of string types, the length is not part of the TypeDescription: it is conveyed as part of the array contents;

  NOTE For example if the data includes octet-strings, only the tag [9] is included in the contents-description. The length of the octet-string is included in the array-contents. With this, string type data with different lengths (including a length of 0) can be encoded in the compact-array.
- in the case of *array*, it includes the tag of the array [1], the number of elements in the array (Unsigned16) and the TypeDescription of the elements in the array;
- in the case of *structure* the TypeDescription is specified as a SEQUENCE OF TypeDescription. Therefore, the contents-description includes the tag of the structure [2], the number of elements in the structure and the TypeDescription of each element in the structure.
- the array-contents includes the series of data – when relevant (in the case of string types) together with its length, without repeating the data type – as an octet string;

Note also that although the contents-description and the array-contents elements of the compact-array type are tagged, these tags do not have to be encoded, as specified in IEC 61334-6:2000, 6.9:

> *A-XDR encoding of a SEQUENCE value shall be the A-XDR encoding of one data value from each of the types listed in the ASN.1 definition of the SEQUENCE type, in the order of their appearance in the definition, unless the type was referenced with the keyword "OPTIONAL" or the keyword "DEFAULT".*
>
> *Tags of explicitly tagged components of a SEQUENCE value represent redundant information, therefore are not encoded: A-XDR encoding of an explicit tagged component value is the A-XDR encoding of the component value.*

F.3.3 Example 1: Compact array encoding an array of five long-unsigned values

An array of 5 elements of type long-unsigned has to be encoded.

The values of the five elements are: 11 11, 22 22, 33 33, 44 44, 55 55.

Encoding as array	Encoding as compact-array
01 // tag of array	13 // tag of compact-array
05 // number of elements	// contents-description
12 11 11 // tag of long-unsigned type and first value	12 // tag of the long-unsigned type
12 22 22 // tag of long-unsigned type and second value	//
12 33 33 // etc.	0 … tet-string
12 44 44	1 … 3 44 44 55 55
12 55 55	//

The length of the encoded data in the two cases is shown in the table below. In the case of the long-unsigned type, 33 % per element can be saved.

	Array	Compact array	Gain compared to array
Header	2	3	-1
First element	3	2	1
Second element	3	2	1
Third element	3	2	1
Fourth element	3	2	1
Fifth element	3	2	1
Total	**17**	**13**	**4**

F.3.4 Example 2: Compact-array encoding of five octet-string values

An array of five octet-string values has to be encoded, of which one is of zero length.

- 31 32 33 34 35 36 37 38
- 41 42 43 44 45 46 47 48
- -
- 31 32 33 34 35 36 37 38
- 41 42 43 44 45 46 47 48

Encoding as array	Encoding as compact-array
01 // tag of array	13 // tag of compact array
05 // number of elements	// contents description
	09 // tag for octet-string
	// array contents
09 08 31 32 33 34 35 36 37 38 // type – length - value	25 // length of octet-string, 37 bytes
09 08 41 42 43 44 45 46 47 48 // second value	08 31 32 33 34 35 36 37 38 // length - value
09 00 // third value, octet-string of length 0	08 41 42 43 44 45 46 47 48 // second length - value
09 08 31 32 33 34 35 36 37 38 // fourth value	00 // third value octet-string of length 0
09 08 41 42 43 44 45 46 47 48 // fifth value	08 31 32 33 34 35 36 37 38 // fourth length - value
	08 41 42 43 44 45 46 47 48 // fifth length - value

The length of the encoded data in the two cases is shown in the table below.

In the case of octet-string, the gain depends on the length of the octet-string. In the case of octet-string of length zero (null-data) the gain is 50 % per element.

	Array	Compact array	Gain compared to array
Header	2	3	-1
First element	10	9	1
Second element	10	9	1
Third element	2	1	1
Fourth element	10	9	1
Fifth element	10	9	1
Total	**44**	**40**	**4**

F.3.5 Example 3: Encoding of the buffer of a Profile generic object

The profile has a time stamp column, a status column, and two columns carrying a double-long-unsigned value each (e.g. A+ and A-, import and export active energy). The capture period is 900 s. There are 96 entries (one day).

NOTE If instead of register readings just delta values would be stored, they could be represented as long-unsigned instead of double-long-unsigned.

Entry	Timestamp	Status	Value	Value	Bytes
1	07D00101FF000000FF800000	80	00000101	00000001	21
2	07D00101FF000F00FF800000	00	00000102	00000002	21
3	07D00101FF001E00FF800000	00	00000103	00000003	21
4	07D00101FF002D00FF800000	00	00000104	00000004	21
…..					…
96	07D00101FF172D00FF800000	00	00000196	00000096	21
				Total bytes	**2 016**

Encoding as array (using the null-data feature)	**Encoding as compact-array**

01 60 // array of 96 elements	13 // compact-array
	// contents-description
	02 04 09110606
	// structure of four elements:
	// octet-string,
	// unsigned,
	// double-long-unsigned,
	// double-long-unsigned

	// array-contents 8203CC // length of octet-string 972 bytes
// first structure, 28 bytes 0204 // structure of 4 elements 090C07D00101FF000000FF800000 1180 // status value is 80 0600000101 0600000001	// first structure, 22 bytes 0C07D00101FF000000FF800000 80 00000101 00000001

Encoding as array (using the null-data feature)	Encoding as compact-array
// second structure, 15 bytes 0204 00 // null-data for time stamp 1100 // status value is 0 0600000102 0600000002	// second structure, 10 bytes 00 // an octet-string of length 0 has the same effect as null-data 00 // status value is 0 00000102 00000002
// third structure 14 bytes 0204 00 // null-data for time stamp 00 // null-data for status 0600000103 0600000003	// third structure, 10 bytes 00 // octet-string of length 0 00 // status value is 0 00000103 00000003
// fourth structure 14 bytes 0204 00 // null-data for time stamp 00 // null-data for status 0600000104 0600000004	// fourth structure, 10 bytes 00 // octet-string of length 0 00 // status value is 0 00000104 00000004
…	…
// ninety-sixth structure 14 bytes 0204 00 // null-data for time stamp 00 // null-data for status 0600000196 0600000096	// ninety-sixth structure, 10 bytes 00 // octet-string of length 0 00 // status value is 0 00000196 00000096
In the case of using compact array, it is not possible to know the number of elements from the length of the octet string of the array-contents. In the case of compact array encoding, the null-data feature can be applied only for string type data.	

The length of the encoded data in the two cases is shown in the table below.

	Array (using null-data)	Compact array	Gain compared to array
Header	2	10	-8
First structure	28	22	6
Second structure	15	10	5
Third structure	14	10	4
Fourth structure	14	10	4
96th structure	14	10	4
Total	**1361**	**982**	**375**
Encoded data in % of raw data	**68%**	**49%**	

When encoding the data as an array, the use of the null data feature allows compression of 32 % in this example. When encoding as a compact array the compression is 51 %.

Annex G
(normative)

NSA Suite B elliptic curves and domain parameters

NOTE This information is reproduced from NSA2.

Domain parameters D for ECC schemes are of the form: $(q, FR, a, b\{, SEED\}, G, n, h)$, where q is the field size; FR is an indication of the basis used; a and b are two field elements that define the equation of the curve; $SEED$ is an optional bit string that is included if the elliptic curve was randomly generated in a verifiable fashion; G is a generating point consisting of (x_G, y_G) of prime order on the curve; n is the order of the point G; and h is the cofactor (which is equal to the order of the curve divided by n).

Suite B requires the use of one of the following two sets of domain parameters, see Table G.1 and Table G.2 :

Table G.1 – ECC_P256_Domain_Parameters

Parameter name	Symbol	Value
Field size	q	FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF $=(2^{224}(2^{32}-1)+2^{192}+2^{96}-1$
Field representation indicator	FR	NULL
Curve parameter	a	FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC
Curve parameter	b	5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
Seed used to generate parameter b:	SEED	C49D3608 86E70493 6A6678E1 139D26B7 819F7E90
x-coordinate of base point G	x_G	6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
y-coordinate base point G	y_G	4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5
Order of point G	n	FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
Cofactor	h	1

Table G.2 – ECC_P384_Domain_Parameters

Parameter name	Symbol	Value
Field size	q	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFF $=2^{384}-2^{128}-2^{96}+2^{32}-1$
Field representation indicator	FR	NULL
Curve parameter	a	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFC
Curve parameter	b	B3312FA7 E23EE7E4 988E056B E3F82D19 181D9C6E FE814112 0314088F 5013875A C656398D 8A2ED19D 2A85C8ED D3EC2AEF
Seed used to generate parameter b:	SEED	A335926A A319A27A 1D00896A 6773A482 7ACDAC73
x-coordinate of base point G	x_G	AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98 59F741E0 82542A38 5502F25D BF55296C 3A545E38 72760AB7
y-coordinate base point G	y_G	3617DE4A 96262C6F 5D9E98BF 9292DC29 F8F41DBD 289A147C E9DA3113 B5F0B8C0 0A60B1CE 1D7E819D 7A431D7C 90EA0E5F
Order of point G	n	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF C7634D81 F4372DDF 581A0DB2 48B0A77A ECEC196A CCC52973
Cofactor	h	1

Annex H
(informative)

Example of an End entity signature certificate using P-256 signed with P-256

Version	3
Serial Number	71
Signature Algorithm	ecdsa-with-SHA256
Issuer	O=DLMS-PKI,CN=SUB-CA
Validity	Not Before: Jan 1 00:00:00 1970 GMT
	Not After: Dec 31 23:59:59 9999 GMT
Subject	CN=MMM12345678
Public Key Algorithm	id-ecPublicKey
Public-Key	Pub:04:f5:44:80:11:79:bb:4e:30:86:95:2b:8d:e4:8e:ba:79:57: cf:19:ad:5b:d3:f7:ec:b1:31:bf:71:9a:1c:2f:e7:8a:93:48:d7:66: d0:67:c5:fb:c1:4b:25:8a:03:a4:6a:b0:f0:0a:09:9f:88:7e:6a:d2: 20:05:e6:03:9b:70:1e
ASN1 OID	prime256v1
Authority Key Identifier	keyid:9D:BA:19:85:19:73:DA:7E:C7:71:55:B2:30:EF:A1:BD:F5:DA:80:F9
Key Usage	Critical, Digital Signature
Signature Algorithm: ecdsa-with-SHA256	30:44:02:20:1b:87:dd:69:07:8a:73:22:7e:2f:43:ba:7c:b0: e5:13:9d:f2:aa:6b:f8:7c:ea:83:e2:fc:09:8f:e9:60:99:d6: 02:20:28:d7:0c:bc:cf:45:24:46:ab:e2:58:2e:a4:94:05:d9: 7b:2e:79:57:c9:3c:40:4f:d0:49:39:2b:e7:db:a0:63

Field	Value	Comments
begin tbsCertificate		
version	2	X.509 version 3
serialNumber	INTEGER	Positive, 20 octets or less
signature	1.2.840.10045.4.3.2	ECDSA with SHA-256
validity		Follows RFC5280
subject		Follows RFC5280, if empty, the subjectAltName must be present and Critical
Unique Identifiers		
subjectUniqueID	Bit string	Optional
subjectPublicKeyInfo		
AlgorithmIdentifier		
algorithm	1.2.840.10045.2.1	EC
parameters	1.2.840.10045.3.1.7	P-256 named curve
subjectPublicKey	bit string 528 bits	1st byte = 0, 2nd byte = 4 (uncompressed) 256 bit x, 256 bit y coordinates
Extensions		
Authority Key Identifier Identifier Value Critical	 2.5.29.35 Octet String False	 Follows RFC5280 8 or 20 octets
Key Usage Identifier Value critical	 2.3.29.15 DER encoded bit string True	0 digitalSignature 4 keyAgreement 5 keyCertSign 6 cRLSign At least one bit must be 1
subjectAltName Identifier Value Critical	 2.5.29.17 OID(s) True when CN is absent	
CertificatePolicies Identifier Value critical	 2.5.29.32 OID Depends on companion spec.	
Subject Key Identifier Identifier Value Critical	 2.5.29.14 Octet String False	Follows RFC5280, applicable to CAs 8 or 20 octets
end tbsCertificate		

signatureAlgorithm	1.2.840.10045.4.3.2	ECDSA with SHA-256
signatureValue	Bit string	Encoded bit string value of a DER encoded sequence of 2 integers; each a maximum of 33 bytes.

Annex I
(normative)

Use of key agreement schemes in DLMS®/COSEM

I.1 Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

Figure I. 1 shows how the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme – specified in 5.3.4.6.2 – is used in DLMS®/COSEM, by invoking the appropriate methods of the "Security setup" IC. See also 5.5.5.

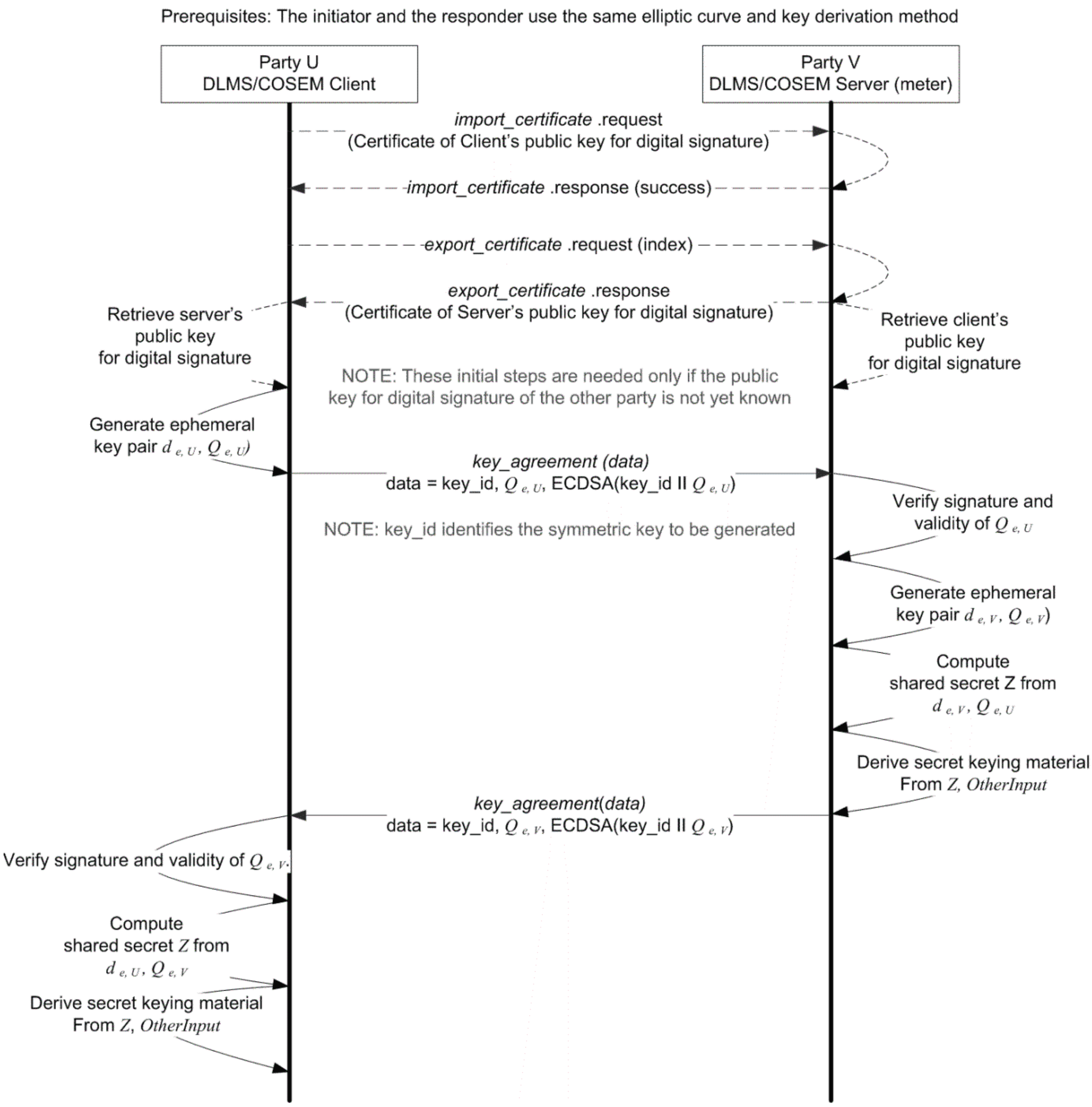


Figure I. 1 – MSC for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

The steps are the following (for details, please refer to NIST SP 800-56A Rev. 2: 2013, 6.1.2.2 and NSA2 3.1):

- Step 1 (optional): the client sends to the server the certificate of its public key for digital signature by invoking the *import_certificate* method;
- Step 2 (optional): the client retrieves from the server the certificate of the public key for digital signature by invoking the *export_certificate* method;
- Step 3: The client generates an ephemeral key pair ($d_{e,\ U}$, $Q_{e,\ U}$). It signs (key_Id, $Q_{e,\ U}$) with its private digital signature key and sends it to the server by invoking the *key_agreement* method;
- Step 4: The server verifies the signature and the validity of $Q_{e,\ U}$. It computes shared secret Z from ($d_{e,\ v}$, $Q_{e,\ U}$) and derives the secret key from Z and $OtherInput;$
- Step 5: If the key has been successfully derived the server generates then an ephemeral key pair ($d_{e\ ,v}$, $Q_{e\ ,v}$). It signs (key_id, $Q_{e\ ,v}$) and sends it to the client in the response to the invocation of the *key_agreement* method;
- Step 5: The client computes shared secret Z from ($d_{e,\ U}$, $Q_{e,\ V}$) and derives the secret key from Z and $OtherInput.$

Table I. 2 provides a test vector.

Table I. 1 – Test vector for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

Security material	**Symbol**	**Contents**	**LEN(X) Bytes**	**LEN(X) Bits**
Security Suite		ECDH-ECDSA-AES-GCM-128-SHA-256		
Curve		P-256		
Domain Parameters	*D*	See Annex G.		
System Title Client	Sys-TC	4D4D4D0000BC614E	8	64
System Title Server	Sys-TS	4D4D4D0000000001	8	64
Private Signing Key Client	Pri-KC	418073C239FA6125011DE4D6CD2E645780289F761BB21 BFB0835CB5585E8B373	32	256
Private Signing Key Server	Pri-KS	AE55414FFE079F9FC95649536BD1C2B5653D200813727 E07D501A8B550C69207	32	256
Public Signing Key Client	Pub-KC	BAAFFDE06A8CB1C9DAE8D94023C601DBBB249254BA22E DD827E820BCA2BCC64362FBB83D86A82B87BB8B7161D2 AAB5521911A946B97A284A90F7785CD9047D25	64	512
Public Signing Key Server	Pub-KS	933ACF15B03A9248E029B2787FB52A0AECAF635F07C42 A0019FB3197E38F8F549A125EA36781B0CA96BE89A0E1 FE2CF9B7361ED48B3C5E24592B9C0F4EDD31D1	64	512
Ephemeral Public Key Client	Epub-KC	2914D60E10AB705F62ED6CC349D7CB99B9AB3F3978E59 278C7AF595B3AF987941372DAB6D5AF1FA867E134167E 6F23DE664A6693E05F43414611058D1B48F894	64	512
Ephemeral Public Key Server	Epub-KS	95F41066009B185B074F5FFFF736B71C325FCADB2BC0C F1A4F4B17BBE7AB81D62946506BC8169C7B539B39A5D8 463787F449C9BD2583FA67A1075B0DBFC638BA	64	512
Ephemeral Private Key Client	Epri-KC	1BAC19FC1D52A1E5102622EDFA36584C05E12FA8CDEAA 450F2F1E9A7DCCF7628	32	256
Ephemeral Private Key Server	Epri-KS	34A8C23A34DBB519D09B245754C85A6CFE05D14A063EF A5AA41545AA8241EFAE	32	256

Security material	**Symbol**	**Contents**	**LEN(X) Bytes**	**LEN(X) Bits**
Ephemeral Public Key Signature Client	Epub-K-Sig-C	06F0607702AA0E2435A183E2F6B1ECD19629712E389A2 13610C03F77B2590860EA840AF5C3FA1F2BCDF055D474 4E9A01CE9A0E55026BCAA4EEBEB764CED64BB3 // The key_id || Epub-KC are included in signature	64	512
Ephemeral Public Key Signature Server	Epri-K-Sig-S	A92995225CEE004ED4376057EEE9536E97EE6F5BAE43E 59BDBBD515A89FB2CB83F2A270871A31B09338DCF0D17 97466087908BA4A6ED8FD48B9EA067DA67DC4D // The key_id || Epub-KS are included in signature	64	512

key_agreement(data) Client	ACTION-Request	C30140
		0040 00002B0000FF 03 // method id
		01 // optional flag
		0101 // array of 1
		0202 // structure of 2
		1600 // key_id = 0, global unicast encryption key
		098180
		2914D60E10AB705F62ED6CC349D7CB99B9AB3F3978E59
		278C7AF595B3AF987941372DAB6D5AF1FA867E134167E
		6F23DE664A6693E05F43414611058D1B48F894
		// ephemeral public key client 64 bytes
		06F0607702AA0E2435A183E2F6B1ECD19629712E389A2
		13610C03F77B2590860EA840AF5C3FA1F2BCDF055D474
		4E9A01CE9A0E55026BCAA4EEBEB764CED64BB34
		// ephemeral public key signature client 64 bytes
			150	1200

global_key_agreement(data) Server	ACTION-Response	C70140
		00 // success
		01 // optional Get-Data-result present
		00 // data CHOICE
		0101 // array of 1
		0202 // structure of 2
		1600 // key id = 0
		098180 // octet string 128 bytes
		95F41066009B185B074F5FFFF736B71C325FCADB2BC0C
		F1A4F4B17BBE7AB81D62946506BC8169C7B539B39A5D8
		463787F449C9BD2583FA67A1075B0DBFC638BA
		// ephemeral public key server 64 bytes
		A92995225CEE004ED4376057EEE9536E97EE6F5BAE43E
		59BDBBD515A89FB2CB83F2A270871A31B09338DCF0D17
		97466087908BA4A6ED8FD48B9EA067DA67DC4D
		// ephemeral public key signature server 64 bytes
			143	1144

Shared Secret	Z	C1CF8FE7891AEF3617D7190795E61FE6C24EFC3CCA2E08469 BAD1A225CE6EA08	32	256
AlgorithmID	AlgID	60857405080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC,Sys-TS)	KDF	C5F4512846EDE51CFB8CCF59F08A694E002EDF66B4CB1 739AC26A74E49712F46	32	256
Global Unicast Encryption Key	GUEK	C5F4512846EDE51CFB8CCF59F08A694E	16	128

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
NOTE The values of the public keys are represented here as FE2OS(xp)II FE2OS(yp).				

I.2 One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme

Figure I. 2 shows how the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme, specified in 5.3.4.6.3 is used in DLMS®/COSEM to protect an xDLMS APDU. See also 5.5.5.

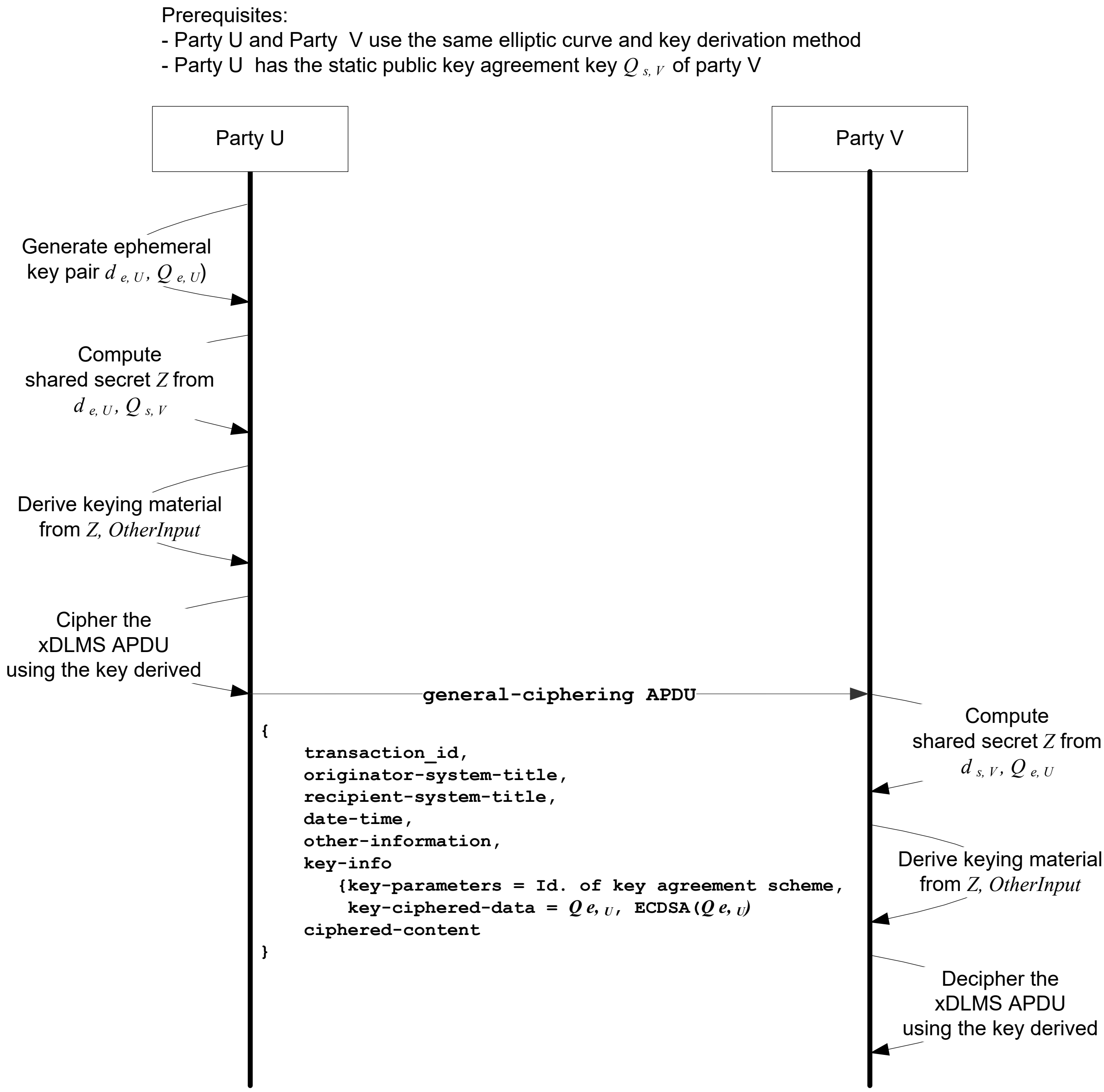


Figure I. 2 – Ciphered xDLMS APDU protected by an ephemeral key established using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme

The process is the following:

- Step 1: The originator, taking the role of the party U of the key agreement process generates an ephemeral key pair ($d_{e,\ U}$, $Q_{e,\ U}$);
- Step 2: It computes shared secret Z from $d_{e,\ U}$, $Q_{s,\ V}$;
- Step 3: It derives the secret key from Z and *OtherInfo*;
- Step 4: It ciphers the xDLMS APDU as required by the security policy in force and by the access rights, using the key derived;
- Step 5: It sends a general-ciphering APDU to the recipient. The use of the fields of the APDU shall be as follows (see also 5.3.4.6.5):
  – transaction-id: as required; not needed for the key derivation process;
  – originator-system-title: this is used as the *PartyUInfo* element of *OtherInfo;*
  – recipient-system-title: this is used as the *PartyVInfo* element of *OtherInfo;*
  – date-time: as required; not needed for the key derivation process;
  – other-information: as required; not needed for the key derivation process;
  – key-info:
  - key-parameters: Identifier of the key agreement scheme: 0x01, see Table 11;
  - key-ciphered-data = $Q_{e,\ U}$ signed by the digital signature private key of Party U;
  – ciphered-content: carries the ciphered xDLMS APDU that is protected using the key.
- Step 6: The recipient, taking the role of party V of the key agreement process computes shared secret Z from $d_{s,\ V}$, $Q_{e,\ U}$;
- Step 7: It derives the secret key from Z and *OtherInfo*;
- Step 8: It deciphers the xDLMS APDU using the key derived.

Table I. 2 provides a test vector.

Table I. 2 – Test vector for key agreement using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Security Suite		ECDH-ECDSA-AES-GCM-128-SHA-256		
Curve		P-256		
Domain Parameters	*D*	See Annex G.		
System Title Client	Sys-TC	4D4D4D0000BC614E	8	64
System Title Server	Sys-TS	4D4D4D0000000001	8	64
Private Signing Key Client	Pri-KC	418073C239FA6125011DE4D6CD2E6[4]5780289F761BB21 BFB0835CB5585E8B373	32	256
Private Signing Key Server	Pri-KS	AE55414FFE079F9FC95649536BD1C2B5653D200813727 E07D501A8B550C69207	32	256
Public Signing Key Client	Pub-KC	BAAFFDE06A8CB1C9DAE8D94023C601DBBB249254BA22E DD827E820BCA2BCC64362FBB83D86A82B87BB8B7161D2 AAB5521911A946B97A284A90F7785CD9047D25	64	512
Public Signing Key Server	Pub-KS	933ACF15B03A9248E029B2787FB52A0AECAF635F07C42 A0019FB3197E38F8F549A125EA36781B0CA96BE89A0E1 FE2CF9B7361ED48B3C5E24592B9C0F4EDD31D1	64	512
Private Key Agreement Key Client	Pri-AKC	A51C16FF5C498FCC89323D4A9267CD71BF81FD6F6A891 CD240DA7F3D6F283E65	32	256

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Private Key Agreement Key Server	Pri-AKS	AAD3FD0732E991CF52A74C66C1F2827DDC53522A2E0A1 69D7C4FFCC0FB5D6A4D	32	256
Public Key Agreement Key Client	Pub-AKC	07C56DE2DCAF0FD793EF29F019C89B4A0CC1E001CE94F 4FFBE10BC05E7E66F7671A13FBCF9E662B9826FFF6A69 38546D524ED6D3405F020296BDE16B04F7A7C2	64	512
Public Key Agreement Key Server	Pub-AKS	A653565B0E06070BAE9FBE140A5D2156812AEE2DD5250 53E3EFC850BF13BFDFFCB240BC7B77BFF5883344E7275 908D2287BEFA3725017295A096989D2338290B	64	512
Ephemeral Public Key Client	Epub-KC	C323C2BD45711DE4688637D919F92E9DB8FB2DFC213A8 8D21C9DC8DCBA917D8170511DE1BADB360D50058F794B 0960AE11FA28D392CFF907A62D13E3357B1DC0	64	512
Ephemeral Public Key Server	Epub-KS	6439724714B47CD9CB988897D8424AB946DCD083D37A9 54637616011B9C2378773295F0F850D8DAFD1BBE9FE66 6E53E4F097CD10B38B69622152724A90987444	64	512
Ephemeral Private Key Client	Epri-KC	47DAB03842E5B6E74828EF4F449B378D7DD1A5DAE1FFC A5AE0B0BE0AD18EC57E	32	256
Ephemeral Private Key Server	Epri-KS	819B1BEACC955E29139E368BF4119C126FF799EE16BCB A3F45C1EF16749BCB95	32	256
Ephemeral Public Key Signature Client	Epub-K-Sig-C	B51BE089D0B682863B2217201E73A1A9031968A9B4121 DCBC3281A69739AF87429F5B3AC5471E7B6A04A2C0F2F 8A25FD772A317DF97FC5463FEAC248EB8AB8BE // Epub-KC is included in signature	64	512
Ephemeral Public Key Signature Server	Epri-K-Sig-S	E1FF47974A1F6931A6502F58147463F0E8CC517D47F55 B0AC56DD8AC5C9D0E481934F2D90F9893016BD82B6E3F FE21FF1588F3278B4E9D98EB4FB62ADD64B380 // Epub-KS are included in signature	64	512
general-ciphering (Access-Request)	GC-C	DD 080102030405060708 // transaction-id 084D4D4D0000BC614E // originator-system-title 084D4D4D0000000001 // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0101 // key-parameters 8180C323C2BD45711DE4688637D919F92E9DB8FB2DFC213A8 8D21C9DC8DCBA917D8170511DE1BADB360D50058F794B0960 AE11FA28D392CFF907A62D13E3357B1DC0B51BE089D0B6828 63B2217201E73A1A9031968A9B4121DCBC3281A69739AF874 29F5B3AC5471E7B6A04A2C0F2F8A25FD772A317DF97FC5463 FEAC248EB8AB8BE // key-ciphered-data 81EB3100000000F435069679270C5BF4425EE5777402A6C8D 51C620EED52DBB188378B836E2857D5C053E6DDF27FA87409 AEF502CD9618AE47017C010224FD109CC0BEB21E742D44AB4 0CD11908743EC90EC8C40E221D517F72228E1A26E827F43DC	401	3208

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
		18ED27B5F458D66508B05A2A4CC6FED178C881AFC3BC67064 689BE8BB41C80ABB3C114A31F4CB03B8B64C7E0B4CE77B239 9C93347858888F92239713B38DF01C4858245827A92EF3341 72EA636B31CBBDF2A96AD5D035F66AA38F1A2D97D4BBA9962 2E6B5F18789CECB2DFB3937D9F3E17F8B472098E6563238F3 7528374809836002AEA6E7012D2ADFAA7 // ciphered-content		
general-ciphering(Access-Response)	GC-S	DD 080123456789012345 // transaction-id 084D4D4D0000000001 // originator-system-title 084D4D4D0000BC614E // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0101 // key-parameters 81806439724714B47CD9CB988897D8424AB946DCD083D 37A954637616011B9C2378773295F0F850D8DAFD1BBE9 FE666E53E4F097CD10B38B69622152724A90987444E1F F47974A1F6931A6502F58147463F0E8CC517D47F55B0A C56DD8AC5C9D0E481934F2D90F9893016BD82B6E3FFE2 1FF1588F3278B4E9D98EB4FB62ADD64B380 // key-ciphered-data 3D3100000000B3FFCAA594642D8319CEC6B2A233E2BF4 621D6991B97E4565B986E8CCBE9A299D8E7869723638F F6BB20E66E175E6F2D762CFD26B3D58733 // ciphered-content	226	1808
Shared Secret GC-C	Z-GC-C	0D4385BA0DD756CBCAB9887EB538396EE8F090A14C1079B43 59F115B977F4615	32	256
AlgorithmID GC-C	AlgID-GC-C	60857405080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC, Sys-TS) GC-C	KDF-GC-C	59A71FD81C929A86A99438DA17A66C058C6A93FD3065F 5EE16A05D775927659B	32	256
Encryption Key GC-C	EK-GC-C	59A71FD81C929A86A99438DA17A66C05	16	128
Shared Secret GC-S	Z-GC-S	2B4302DC49790E2E78D990CFB52ED6E2F273DECE441A2D95E 4301B93812A9FAC	32	256
AlgorithmID GC-S	AlgID-GC-S	60857405080300	7	56
KDF(Z,AlgID,Sys-TS, Sys-TC) GC-S	KDF-GC-S	F0184BDA9466BFA4601A64A7EF46504AB1A40C4851A0A6445 03599DF298B2E14	32	256
Encryption Key GC-S	EK-GC-S	F0184BDA9466BFA4601A64A7EF46504A	16	128
NOTE The values of the public keys are represented here as FE2OS(xp)II FE2OS(yp).				

I.3 Static Unified Model C(0e, 2s, ECC CDH) scheme

Figure I. 3 shows how Static Unified Model C(0e, 2s, ECC CDH) schemes specified in 5.3.4.6.4 is used in DLMS®/COSEM to protect an xDLMS APDU. See also 5.5.5.

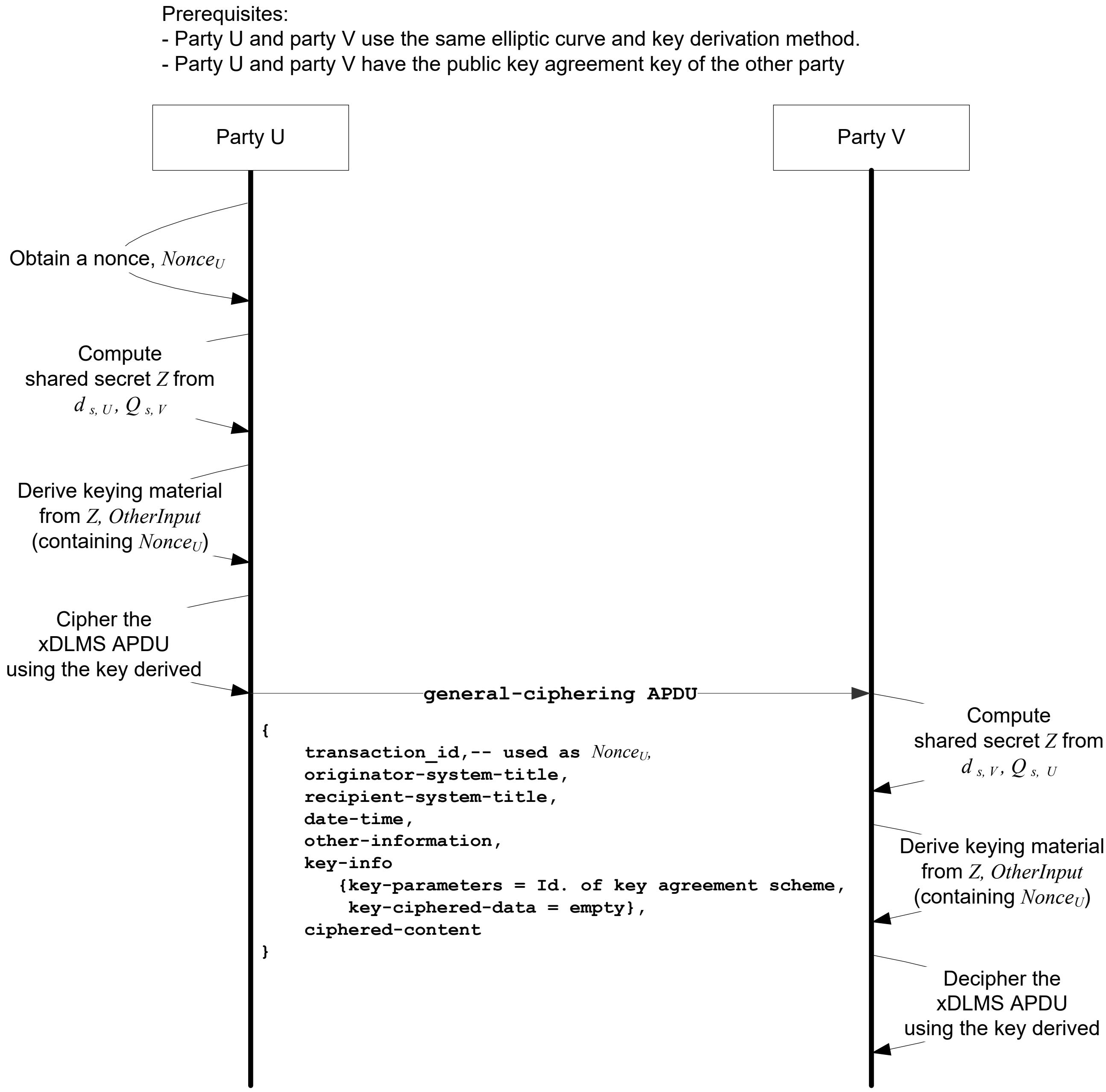


Figure I. 3 – Ciphered xDLMS APDU protected by an ephemeral key established using the Static Unified Model C(0e, 2s, ECC CDH) scheme

The process is the following:

- Step 1: The originator, taking the role of the Party U of the key agreement process obtains a nonce, $Nonce_U$;
- Step 2: It computes shared secret Z from $d_{s,\ U}$, $Q_{s,\ V}$;

NOTE See also Note to Table 6.

- Step 3: It derives the secret key from Z, and $OtherInput$ that contains $Nonce_U$;
- Step 4: It ciphers the xDLMS APDU APDU as required by the security policy in force and by the access rights, using the key derived;
- Step 5: It sends a general-ciphering APDU to the recipient. The use of the fields of the APDU shall be as follows (see also 5.3.4.6.5):
  - transaction-id: this field is used as $Nonce_U$;
  - originator-system-title: this is used as the $PartyUInfo$ element of $OtherInfo$;
  - recipient-system-title: this is used as the $PartyVInfo$ element of $OtherInfo$;
  - date-time: as required; not needed for the key derivation process;
  - key-info:
  - key-parameters: Identifier of the key agreement scheme: 0x02, see Table 11.
  - key-ciphered-data = empty;
  - ciphered-content carries the ciphered xDLMS APDU that is protected using the key.
- Step 6: The recipient, taking the role of party V of the key agreement process computes shared secret Z from $d_{s,\ V}$, $Q_{s,\ U}$;
- Step 7: It derives the secret key from Z and $OtherInput$ that contains $Nonce_U$;
- Step 8: It deciphers the xDLMS APDU using the key derived.

Table I. 3 provides a test vector.

Table I. 3 – Test vector for key agreement using the Static-Unified Model (0e, 2s, ECC CDH) scheme

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Security Suite		ECDH-ECDSA-AES-GCM-128-SHA-256		
Curve		P-256		
Domain Parameters	*D*	See Annex G.		
System Title Client	Sys-TC	4D4D4D0000BC614E	8	64
System Title Server	Sys-TS	4D4D4D0000000001	8	64
Private Key Agreement Key Client	Pri-AKC	A51C16FF5C498FCC89323D4A9267CD71BF81FD6F6A891 CD240DA7F3D6F283E65	32	256
Private Key Agreement Key Server	Pri-AKS	AAD3FD0732E991CF52A74C66C1F2827DDC53522A2E0A1 69D7C4FFCC0FB5D6A4D	32	256
Public Key Agreement Key Client	Pub-AKC	07C56DE2DCAF0FD793EF29F019C89B4A0CC1E001CE94F 4FFBE10BC05E7E66F7671A13FBCF9E662B9826FFF6A69 38546D524ED6D3405F020296BDE16B04F7A7C2	64	512
Public Key Agreement Key Server	Pub-AKS	A653565B0E06070BAE9FBE140A5D2156812AEE2DD5250 53E3EFC850BF13BFDFFCB240BC7B77BFF5883344E7275 908D2287BEFA3725017295A096989D2338290B	64	512
Nonce Client	Nonce-C	080102030405060708 // Nonce-C is transaction-id with length	9	72
Nonce Server	Nonce-S	080123456789012345 // Nonce-S is transaction-id with length	9	72

Security material	**Symbol**	**Contents**	**LEN(X) Bytes**	**LEN(X) Bits**
general-ciphering (Access-Request)	GC-C	DD 080102030405060708 // transaction-id 084D4D4D0000BC614E // originator-system-title 084D4D4D0000000001 // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0102 // key-parameters 00 // key-ciphered-data not present 81EB3100000000607581D83815281B561904E6A72B83BF3FB0B1F2A0A23A82A804B39911F6CB1B4EF9B6F76C6338ED058014FFBF4A13A162E0EED11EEB8F597757A18215F28E1D3FC2D44586C4B8AFE4500B535B579506CDB925CBEFF7F1CF6BF96C583B9CF588FE8F6B01A574824D7CEC597F057FFA8700AF12AD63A7FA72040439F4392C089B265F9EB1AC308239906DC04E1A8712ABE383CF92349842EBA166EF2E9EB2F53D0DBA025D79875463398BBC3007BC4A6FC12780C21EE1ABF080BF0FA926242834C0AC30D55A1EF8856E7DED48621F17FDF4D5B54EDDADD874119251049B6AEE111C12FBC2FEAF // ciphered-content	272	2176
general-ciphering (Access-Response)	GC-S	DD 080123456789012345 // transaction-id 084D4D4D0000000001 // originator-system-title 084D4D4D0000BC614E // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0102 // key-parameters 00 // key-ciphered-data not present 3D31000000003CF971CD09E0B1C5B89E7BB52C1B39923D14C4A160D7DDB3F2DE8AD255C625F407F04031CD95F4F261E23E823B73490CD6CB593A140CD95F ciphered-content	97	776
Shared Secret GC-C	Z-GC-C	B1455B2AD5F68BCFFE6AD5412BA89548ACA7E0CBF4B1560D6A57496F15E931AD	32	256

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
AlgorithmID GC-C	AlgID-GC-C	60857405080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC, Nonce-C, Sys-TS) GC-C	KDF-GC-C	56C46B57DF675515C31025455822514AFA2CDEB3E0BF1 CADA84576159E84DE7E	32	256
Encryption Key GC-C	EK-GC-C	56C46B57DF675515C31025455822514A	16	128
Shared Secret GC-S	Z-GC-S	B1455B2AD5F68BCFFE6AD5412BA89548ACA7E0CBF4B1560D6 A57496F15E931AD	32	256
AlgorithmID GC-S	AlgID-GC-S	60857405080300	7	56
KDF(Z,AlgID,Sys-TS, Nonce-S, Sys-TC) GC-S	KDF-GC-S	FC1314F7DE033B7DD19C80DBCF9FF2C5286DC8F76E87877BD 90B9B7F00CD5613	32	256
Encryption Key GC-S	EK-GC-S	FC1314F7DE033B7DD19C80DBCF9FF2C5	16	128
NOTE The values of the public keys are represented here as FE2OS(xp)II FE2OS(yp).				

Annex J
(informative)

Exchanging protected xDLMS APDUs between TP and server

J.1 General

This use case shows exchanging protected xDLMS APDUs between a third party (TP) and a server via a client.

J.2 Example 1: Protection is the same in the two directions

In the first example, the security policy of the server requires that the request is digitally signed and authenticated and the response is also digitally signed and authenticated.

In the .request, the digital signature is applied by the TP and the authentication is applied by the client:

- the TP sends the .request to the client in a general-signing APDU for the server: the recipient-system-title is that of the server;
- the client verifies the digital signature and if correct, encapsulates the general-signing APDU in a general-ciphering APDU.

The server checks and removes first the authentication and then the digital signature.

If both are correct, it prepares the .response APDU. The protection is applied to the same parties as in the request:

- the server first encapsulates the .response APDU in a general-signing APDU for the TP: the destination-system-title is that of the TP;
- it encapsulates then this general-signing APDU in a general-ciphering APDU for the client: the destination-system-title is that of the client.

The protection to be applied by each party is subject to project specific companion specifications, but the overall protection shall meet the security policy configured in the server. For example, the server would accept any of the following:

– digital signature applied by the TP and authentication applied by the client;
– authentication applied by the TP and digital signature applied by the client;
– both digital signature and authentication applied by the client;
– both digital signature and authentication applied by the TP.

The process is shown in Figure J.1.

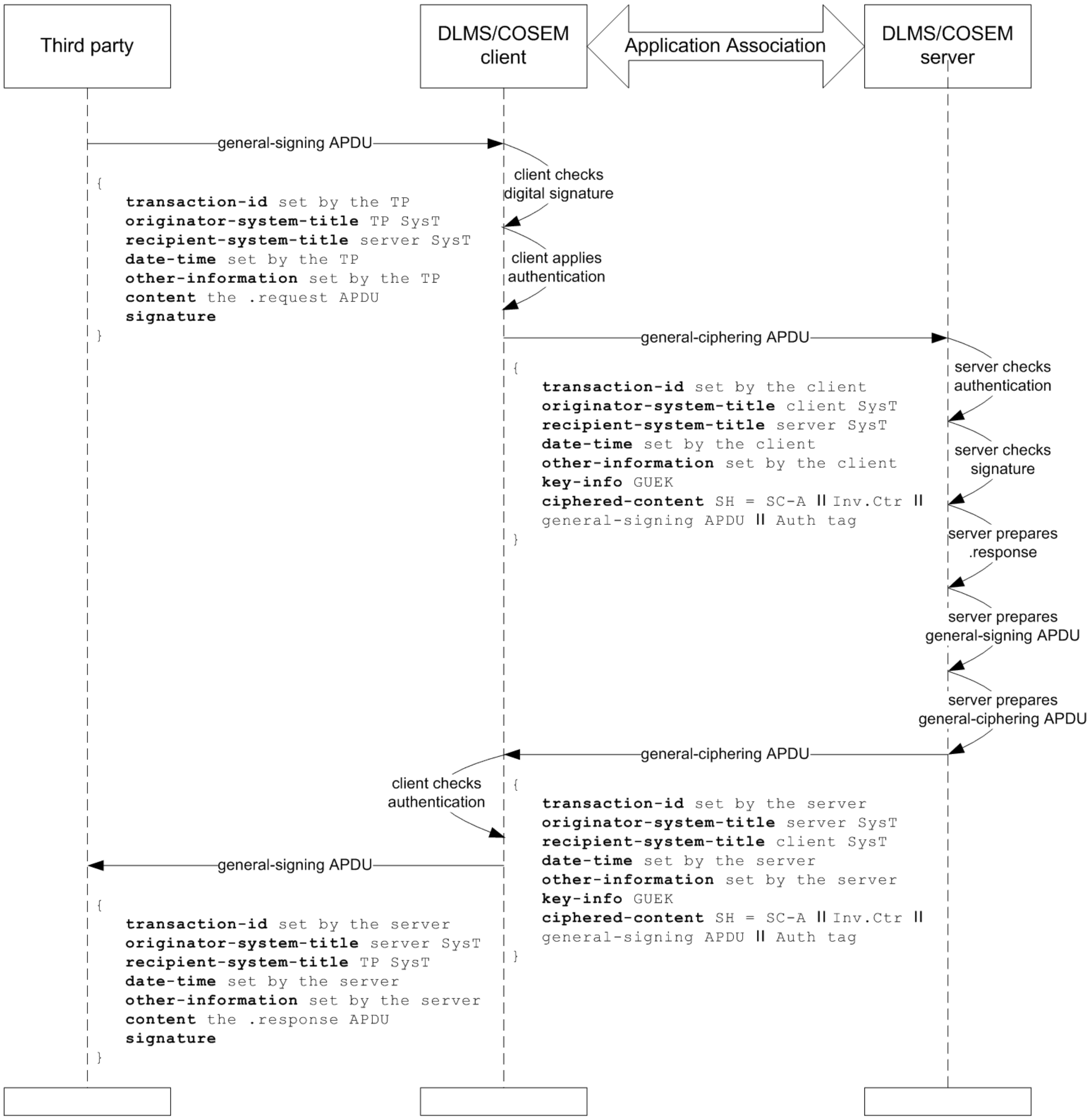


Figure J.1 – Exchanging protected xDLMS APDUs between TP and server: example 1

J.3 Example 2: Protection is different in the two directions

In the second example, the security policy of the server requires that the request is digitally signed and authenticated and the response is only authenticated.

In the .request, the digital signature is applied by the TP and the authentication is applied by the client:

- the TP sends the .request to the client in a general-signing APDU for the server: the recipient-system-title is that of the server;
- the client verifies the digital signature and if correct, encapsulates the general-signing APDU in a general-ciphering APDU.

The server checks and removes first the authentication tag and then the digital signature.

If both are correct, it prepares the .response APDU. The protection is applied to the same parties as in the request:

- the server first encapsulates the .response APDU in a general-ciphering APDU for the TP: the destination-system-title is that of the TP, but no protection is applied: in the Security Control Byte, the bits indicating authentication and encryption are set to 0;
- it encapsulates then this general-ciphering APDU in a general-ciphering APDU for the client: the destination-system-title is that of the client.

The process is shown in Figure J.2.

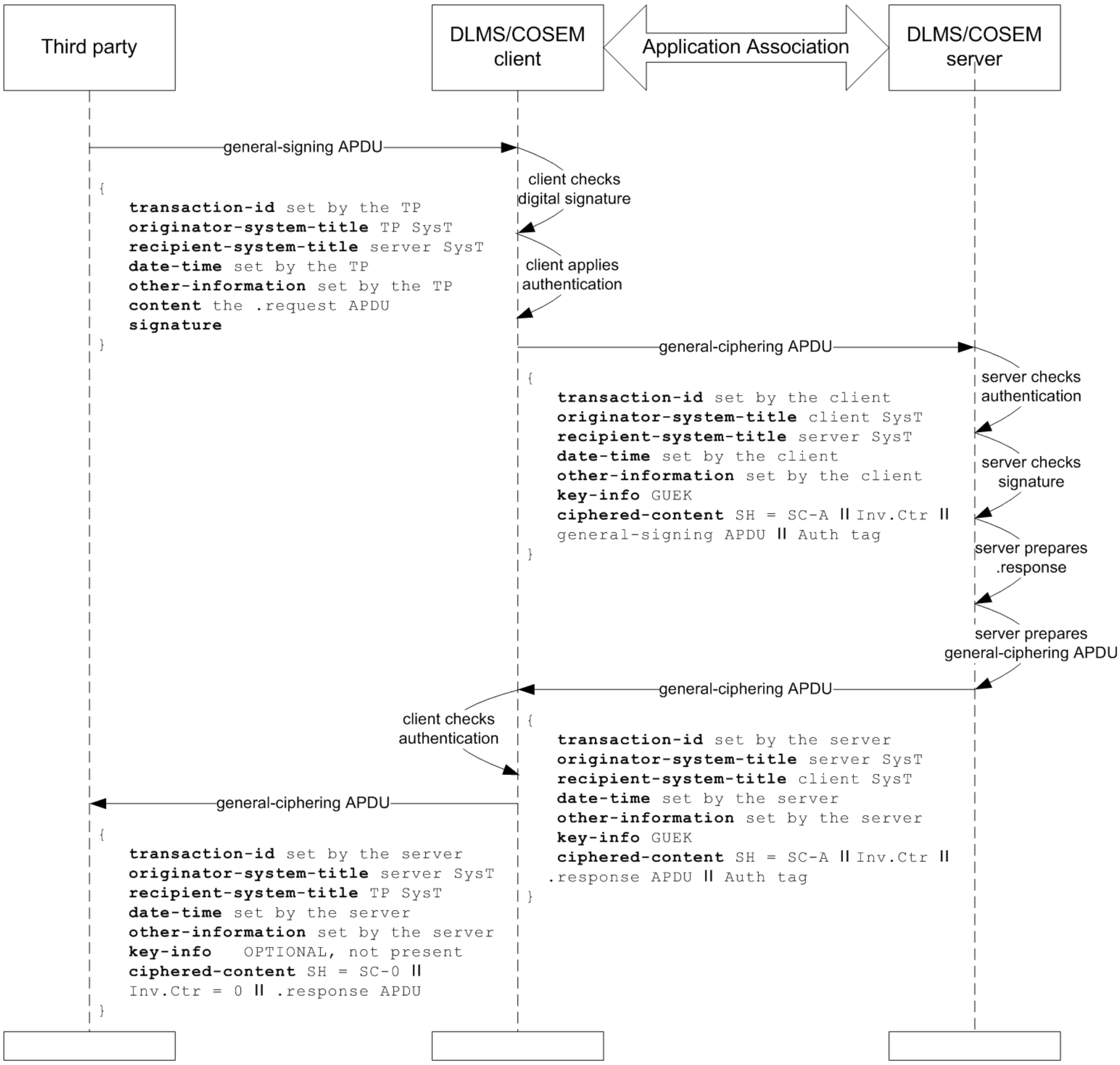


Figure J.2 – Exchanging protected xDLMS APDUs between TP and server: example 2

Annex K
(informative)

Significant technical changes with respect to IEC 62056-5-3:2016

This third edition of IEC 62056-5-3 includes the following significant technical changes with respect to the second edition of IEC 62056-5-3:2016.

Item	Clause	
1.	2	New references added.
2.	3.1	General DLMS®/COSEM definitions added
3.	3.2	Definitions related to cryptographic security added
4.	3.4	General abbreviations: new abbreviations added
5.	3.3	Definitions, abbreviations, symbols and notation relevant for the Galois/Counter Mode brought here from Green Book Ed.7 9.2.4.8.2
6.	3.6	Definitions, abbreviations, symbols and notation relevant for the ECDSA algorithm added.
7.	3.7	Definitions, abbreviations, symbols and notation relevant for the key agreement algorithms added
8.	4.1	Specifies the general concepts of information exchange in DLMS®/COSEM, extending IEC 62056-5-3 Ed.2.0:—, Clause 4.
9.	4.1.1	General: Text updated, key characteristics of DLMS®/COSEM listed here.
10.	4.1.2	Communication model: new text introducing APs, AEs, ASEs, AAs and their relationship. Client/server model is explained here.
11.	4.1.3	Naming and addressing: new text bringing all related elements together, and bringing in the concept of data exchange with third parties. Table of SAPs added.
12.	4.1.3.4	System title: IEC 62056-5-3 Ed. 2:—, 5.4.8.3.4.6 on exchanging system titles brought here.
13.	4.1.3.6	Client user identification added.
14.	4.1.4	Connection oriented operation explained.
15.	4.1.5	Application associations. New text, bringing all related elements together.
16.	4.1.6	Messaging patterns: new text bringing in the concept of push operation.
17.	4.1.7	Data exchange between third parties and DLMS®/COSEM servers: new text.
18.	4.1.8	Communication profiles added.
19.	4.1.9	Model of a DLMS®/COSEM metering system
20.	4.1.10	Model of DLMS®/COSEM servers added.
21.	4.1.11	Model of DLMS®/COSEM clients added.
22.	4.1.12	Interoperability and interconnectivity in COSEM added.
23.	4.1.13	Ensuring interconnectivity: the protocol identification service added.
24.	4.1.14	System integration and meter installation added.
25.	4.2	The whole clause has been revised to present the additional services and mechanisms compared to DLMS® as specified in IEC 61334-4-41:1996.

Item	Clause	
26.	5	Information security in DLMS®/COSEM: The whole clause has been reworked to describe the extended security features of DLMS®/COSEM. Main elements: - the DLMS®/COSEM security concept is presented; - not only xDLMS APDUs but also COSEM data – carried by the APDUs – can be protected; - the protection can be applied not only between clients and servers but also between third parties and servers via clients; - symmetric and public key algorithms are available to provide any combination of authentication, encryption and digital signature; - multiple layers of protection can be applied and verified by multiple entities; - key-transport has been complemented by key agreement; - compression (managed together with symmetric key algorithms) added.
27.	6.2	The COSEM-OPEN service: Service parameters and their use updated to support client user identification and transportation of Certificates.
28.	6.5	Protection and general block transfer parameters: clause reworked to cover the new general protection and general block transfer mechanisms and related parameters.
29.	6.6, 6.7	GET, SET, ACTION service: either the service-specific or the general block transfer mechanism can be used.
30.	6.8	The ACTION service: Text on method invocation parameters – use of null-data – precised.
31.	6.9	The ACCESS service: the new unified service added, with the main features presented.
32.	6.10	The DataNotification service: The new service added.
33.	6.14, 6.15	Read, Write service: Either the service-specific or the general block transfer mechanism can be used.
34.	6.16	UnconfirmedWrite service: General block transfer mechanism can be used.
35.	6.19	Summary of services and LN/SN data transfer service mapping: New services added.
36.	7.1.1, 7.1.2	State definitions: Updated to include the new services and APDUs.
37.	7.2.1	ACSE functional units, services and service parameters: Updated to be in line with ISO/IEC 15953:1999 and ISO/IEC 15954:1999 replacing ISO/IEC 8649 and ISO/IEC 8650.
38.	7.2.2.3	The COSEM authentication mechanism name: New mechanisms names added.
39.	7.2.2.4	Cryptographic algorithm ID-s: Added, these IDs are used in the KDF function.
40.	7.2.3	APDU encoding rules: Encoding of the ACSE APDUs, the xDLMS APDUs and XML.
41.	7.2.4	Protocol for the establishment of confirmed application associations: text amended to include parsing of the new parameters.
42.	7.3.1	Negotiation of services and options – the conformance block: New elements added.
43.	7.3.3, 7.3.4, 7.3.5	Protocol of the GET, SET, and ACTION service: It is specified that either the service-specific or the general block transfer mechanism can be used.
44.	7.3.6	Protocol for the ACCESS service added.
45.	7.3.7	Protocol of the DataNotification service added.
46.	7.3.9, 7.3.10	Protocol of the Read and Write service: It is specified that either the service-specific or the general block transfer mechanism can be used.
47.	7.3.13	Protocol of general block transfer mechanism added.
48.	8	Abstract syntax of the new services and APDUs added.
49.	9	COSEM APDU XML schema added.
50.	Annex B	SMS short wrapper added.
51.	Annex C	Gateway protocol added.
52.	F.2	Example for the ACCESS service added.
53.	F.3	Compact array example added.
54.	Annex G	NSA Suite B elliptic curves and domain parameters added.
55.	Annex H	Certificate examples added.

Item	Clause	
56.	Annex I	Use of key agreement schemes in DLMS®/COSEM added.
57.	Annex J	Exchanging protected xDLMS APDUs between TP and server added.
58.	Bibliography	New references added.

Bibliography

The DLMS® UA “Books” are available for the members of the DLMS® User Association. See www.dlms.com

DLMS UA 1000-1, the “Blue Book” Ed. 14.0: 2020, *COSEM interface classes and OBIS identification system*

DLMS UA 1000-1, the “Blue Book” Ed. 13.0: 2019, *COSEM interface classes and OBIS identification system*

DLMS UA 1000-1, the “Blue Book” Ed. 12.0: 2017, *COSEM interface classes and OBIS identification system*

DLMS UA 1000-1, the “Blue Book” Ed. 11.0: 2013, *COSEM interface classes and OBIS identification system*

DLMS UA 1000-1, the “Blue Book” Ed. 12.0: 2014, *COSEM interface classes and OBIS identification system*

DLMS UA 1000-2, the "Green Book" Ed. 10.0 Amendment 1.0:2021, *DLMS®/COSEM Architecture and Protocols*

DLMS UA 1000-2, the "Green Book" Ed. 10.0:2020, *DLMS®/COSEM Architecture and Protocols*

DLMS UA 1000-2, the "Green Book" Ed. 9.01:2019, *DLMS®/COSEM Architecture and Protocols*

DLMS UA 1000-2, the "Green Book" Ed. 8.3:2017, *DLMS®/COSEM Architecture and Protocols*

DLMS UA 1000-2, the "Green Book" Ed. 7.0:2009, *DLMS®/COSEM Architecture and Protocols*

DLMS UA 1000-2, the "Green Book" Ed. 7.0, Amendment 3: 2013, *DLMS®/COSEM Architecture and Protocols, (cancels and replaces Amendment 1 and 2)*

DLMS UA 1000-2, the “Green Book” Ed. 8.1:2015, *DLMS®/COSEM Architecture and Protocols*

DLMS UA 1001-1, the “Yellow Book”, Ed. 4.0:2007, *DLMS®/COSEM Conformance test and certification process*

DLMS UA 1002, the “White Book”, Ed. 1.0:2003, *COSEM Glossary of terms*

IEC 60050-300, *International Electrotechnical Vocabulary – Electrical and electronic measurements and measuring instruments*

IEC 61334-4-32:1996, *Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 32: Data link layer – Logical link control (LLC)*

IEC 61334-4-511:2000, *Distribution automation using distribution line carrier systems – Part 4-511: Data communication protocols – Systems management – CIASE protocol*

IEC 61334-4-512:2001, *Distribution automation using distribution line carrier systems – Part 4-512: Data communication protocols – System management using profile 61334-5-1 – Management Information Base (MIB)*

IEC 61334-5-1:2001, *Distribution automation using distribution line carrier systems – Part 5-1: Lower layer profiles – The spread frequency shift keying (S-FSK) profile*

IEC 62056-1-0, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 1-0: Smart metering standardisation framework*

IEC TS 62056-1-1, Electricity metering data exchange – The DLMS®/COSEM suite – Part 1-1: Template for DLMS®/COSEM communication profile standards

IEC TR 62056-41:1998, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 41: Data exchange using wide area networks: Public switched telephone network (PSTN) with LINK+ protocol*

IEC TR 62056-51:1998*, Electricity metering – Data exchange for meter reading, tariff and load control – Part 51: Application layer protocols*

IEC TR 62056-52:1998*, Electricity metering – Data exchange for meter reading, tariff and load control – Part 52: Communication protocols management distribution line message specification (DLMS®) server*

IEC 62056-6-1:2021, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 6-1: Object Identification System (OBIS)* [5]

IEC 62056-7-3:2017, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 7-3: Wired and wireless M-Bus communication profiles for local and neighbourhood networks*

IEC 62056-7-6:2013, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 7-6: The 3-layer, connection-oriented HDLC based communication profile*

IEC 62056-9-7:2013, *Electricity metering data exchange – The DLMS®/COSEM suite – Part 9-7: Communication profile for TCP-UDP/IP networks*

Future IEC 62056-8-4:2018, *ELECTRICITY METERING DATA EXCHANGE – THE DLMS®/COSEM SUITE – Part 8-4: Narrow-band OFDM PRIME PLC communication profile for neighbourhood networks*

Future IEC 62056-8-5:2017, *ELECTRICITY METERING DATA EXCHANGE – THE DLMS®/COSEM SUITE – Part 8-5: Narrow-band OFDM G3-PLC communication profile for neighbourhood networks*

ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 8802-2:1998, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 2: Logical link control*

5 To be published simultaneously with this part of IEC 62056.

ISO/IEC 9545:1994, *Information technology – Open Systems Interconnection – Application layer structure*

ISO/IEC 9798-1:2010, *Information technology — Security techniques — Entity authentication — Part 1: General Evaluation of ISO/IEC 9798 Protocols Version 2.0 David Basin and Cas Cremers April 7, 2011*

ISO/IEC 9798-2 Ed. 3:2008, *Information technology — Security techniques — Entity authentication — Part 2: Mechanisms using symmetric encipherment algorithms*

ISO/IEC 9798-3:1998, *Information technology – Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques*

ISO/IEC 10731:1994, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ISO/IEC 13239:2002, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*

ISO/IEC 15945:2002, *Information technology — Security techniques — Specification of TTP services to support the application of digital signatures*

ISO 2110:1989, *Information technology – Data communication – 25-pole DTE/DCE interface connector and contact number assignments*

ITU-T V.24:2000, *List of definitions for interchange circuits between data terminal equipment (DTE) and data circuit-terminating equipment (DCE)*

ITU-T V.25:1996, *Automatic answering equipment and general procedures for automatic calling equipment on the general switched telephone network including procedures for disabling of echo control devices for both manually and automatically established calls*

ITU-T V.25bis:1996, *Synchronous and asynchronous automatic dialling procedures on switched networks*

ITU-T V.28:1993, *Electrical characteristics for unbalanced double-current interchange circuits*

ITU-T V.44: 2000 , *SERIES V: DATA COMMUNICATION OVER THE TELEPHONE NETWORK – Error control – V.44:2000, Data compression procedures*

ITU-T X.211:1995, *Information technology – Open Systems Interconnection – Physical service definition*

ITU-T X.509:2008, *SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY – Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks*

ITU-T X.693 (11/2008) , *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)*

ITU-T X.693 Corrigendum 1 (10/2011), *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) Technical Corrigendum 1*

ITU-T X.694 (11/2008), *Information technology – ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1*

ITU-T X.694 Corrigendum 1 (10/2011), Information technology – ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1 Technical Corrigendum 1

ITU-T X.811:1995, Information technology - Open Systems Interconnection - Security frameworks for open systems: Authentication framework

IEEE 802.1 AE:2006, *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security*

IEEE 802.15.4:2006, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*

EN 13757-2:2004, *Communication system for and remote reading of meters – Part 2: Physical and Link Layer*

FIPS PUB 180-4:2012, Secure hash standard (SHS)

FIPS PUB 186-4:2013, Digital Signature Standard (DSS)

FIPS PUB 197:2001, Advanced Encryption Standard (AES)

FIPS PUB 198:2002, *The Keyed-Hash Message Authentication Code (HMAC)*

FIPS PUB 199:2002, *Standards for Security Categorization of Federal Information and Information Systems*

NIST SP 800-47:2002, *Security Guide for Interconnecting Information Technology Systems*

The Galois/Counter Mode of Operation (GCM) - David A. McGrew, Cisco Systems, Inc. 170, West Tasman Drive, San Jose, CA 95032, mcgrew@cisco.com, John Viega, Secure Software, 4100 Lafayette Center Drive, Suite 100, Chantilly, VA 20151, viega@securesoftware.com, May 31, 2005

RFC 1321, *The MD5 Message-Digest Algorithm.* Edited by R. Rivest (MIT Laboratory for Computer Science and RSA Data Security, Inc.) April 1992 *http://www.ietf.org/rfc/rfc1321.txt*

RFC 1662 *PPP in HDLC-like Framing, 1984, http://www.ietf.org/rfc/rfc1662.txt*

RFC 2104 *HMAC: Keyed-Hashing for Message Authentication, 2004, http://www.ietf.org/rfc/rfc2104.txt*

RFC 2119 *Key words for use in RFCs to Indicate Requirement Levels, 1997, http://www.ietf.org/rfc/rfc2119.txt*

RFC 2315 *PKCS #7, Cryptographic Message Syntax Version 1.5, 1998, https://www.ietf.org/rfc/rfc2315*

RFC 2560 *X.509, Internet Public Key Infrastructure – Online Certificate Status Protocol – OCSP, 1999, http://www.ietf.org/rfc/rfc2560*

RFC 2822 *Internet Message Format, 2001, http://www.ietf.org/rfc/rfc2822*

RFC 2986 PKCS #10 v1.7, Certification Request Syntax Standard, http://www.ietf.org/rfc/rfc2986

RFC 3268. *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS), 2002, http://tools.ietf.org/html/rfc3268*

RFC 3394, Advanced Encryption Standard (AES) Key Wrap Algorithm. Edited by J. Schaad (Soaring Hawk Consulting) and R. Housley (RSA Laboratories) September 2002 http://tools.ietf.org/html/rfc3394

RFC 4106, *The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP) http://www.rfc-editor.org/rfc/rfc4106.txt*

RFC 4108 Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages, 2005, http://www.ietf.org/rfc/rfc4108

RFC 4210, Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), 2008, https://tools.ietf.org/html/rfc5349

RFC 4211. *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF), 2005, http://www.ietf.org/rfc/rfc4211*

RFC 4308. *Cryptographic Suites for IPsec, 2005*, *https://www.google.hu/#q=RFC%204308*

RFC 4835. *Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH),2007, http://tools.ietf.org/html/rfc4335*

RFC 5084, *Internet Engineering Task Force (IETF). Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS). Edited by R. Housley November 2007. Available from: http://www.rfc-editor.org/rfc/rfc5084.txt*

RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008, http://www.ietf.org/rfc/rfc5280

RFC 5349, *Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos* (PKINIT), 2008, https://tools.ietf.org/html/rfc5349

ANSI C12.21:1999, *Protocol Specification for Telephone Modem Communication*

SEC1:2009, *Standards for Efficient Cryptography: Elliptic Curve Cryptography.* SECG. Version 2.0

SEC2:2010, *Standards for Efficient Cryptography: Recommended Elliptic Curve Domain Parameters,* Version 2.0. Certicom Research

NIST SP 800-21:2005, Guideline for Implementing Cryptography in the Federal Government

NIST SP 800-32:2001, Introduction to Public Key Technology and the Federal PKI Infrastructure

NIST SP 800-38D:2007, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC

NIST SP 800-56A Rev. 2: 2013, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography

NIST SP 800-57:2012, Recommendation for Key Management – Part 1: General (Revision 3)

NSA1, Suite B Implementer's Guide to FIPS 186-3 (ECDSA), Feb 3rd 2010

NSA2, Suite B Implementer's Guide to NIST SP800-56A, 28th July 2009

NSA3, NSA Suite B Base Certificate and CRL Profile, 27th May 2008

Index