Information security in DLMS/COSEM

This subclause describes and specifies:

• the DLMS/COSEM security concept,
• the cryptographic algorithms selected,
• the security keys,
• the use of the cryptographic algorithms for entity authentication, xDLMS APDU protection and COSEM data protection.

The DLMS/COSEM security concept

The resources of DLMS/COSEM servers – COSEM object attributes and methods – can be accessed by DLMS/COSEM clients within Application Associations

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.

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

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

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

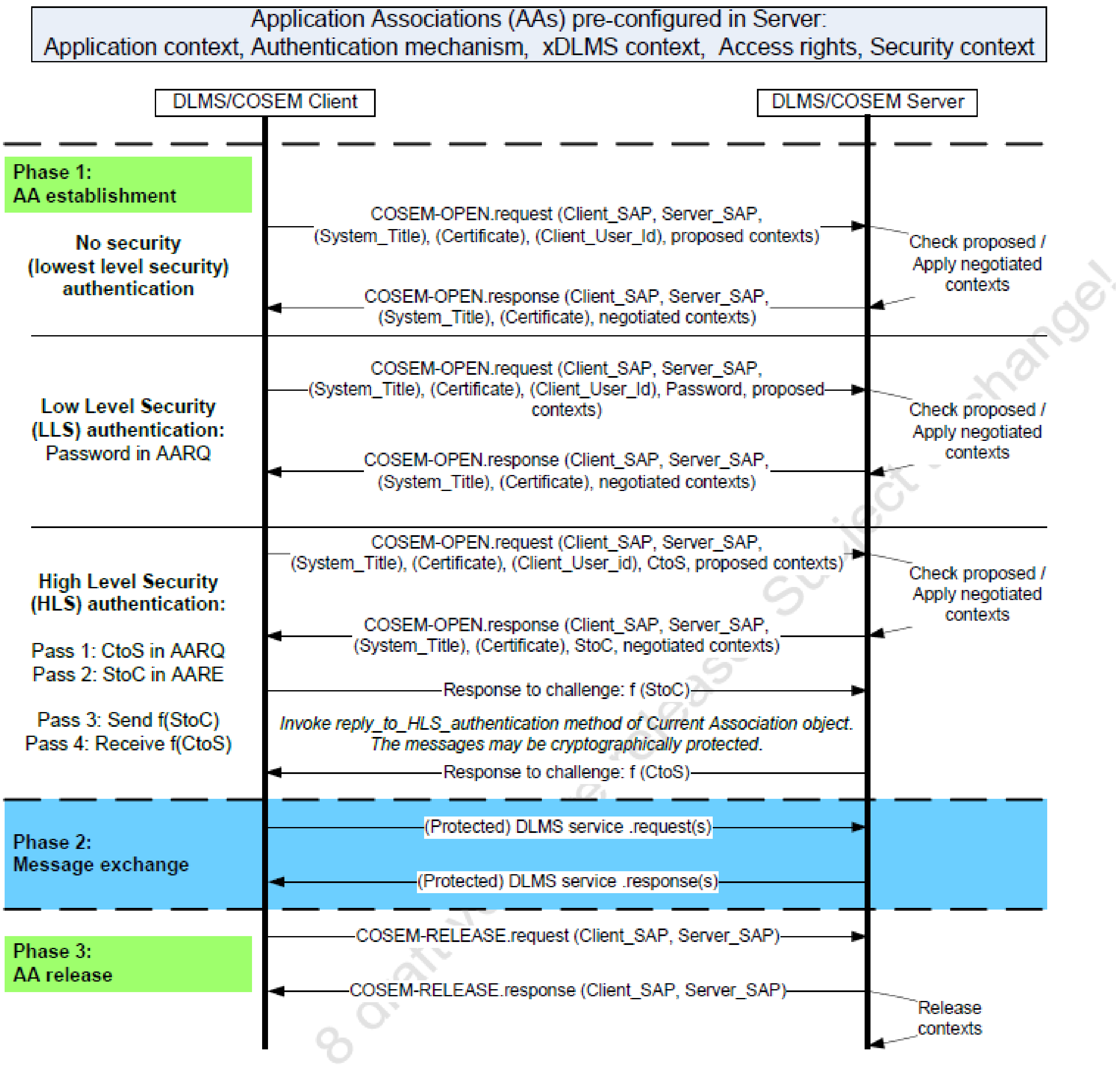
- **Identification and authentication**

- **Identification**

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 and to log their activities accessing the meter

- **Authentication mechanisms**


Application Associations (AAs) pre-configured in Server:
Application context, Authentication mechanism, xDLMS context, Access rights, Security context
DLMS/COSEM Client
DLMS/COSEM Server
Phase 1: AA establishment
No security (lowest level security) authentication
COSEM-OPEN.request (Client_SAP, Server_SAP, (System_Title), (Certificate), (Client_User_Id), proposed contexts)
Check proposed / Apply negotiated contexts
COSEM-OPEN.response (Client_SAP, Server_SAP, (System_Title), (Certificate), negotiated contexts)
Low Level Security (LLS) authentication: Password in AARQ
COSEM-OPEN.request (Client_SAP, Server_SAP, (System_Title), (Certificate), (Client_User_Id), Password, proposed contexts)
Check proposed / Apply negotiated contexts
COSEM-OPEN.response (Client_SAP, Server_SAP, (System_Title), (Certificate), negotiated contexts)
High Level Security (HLS) authentication:
Pass 1: CtoS in AARQ
Pass 2: StoC in AARE
Pass 3: Send f(StoC)
Pass 4: Receive f(CtoS)
COSEM-OPEN.request (Client_SAP, Server_SAP, (System_Title), (Certificate), (Client_User_id), CtoS, proposed contexts)
Check proposed / Apply negotiated contexts
COSEM-OPEN.response (Client_SAP, Server_SAP, (System_Title), (Certificate), StoC, negotiated contexts)
Response to challenge: f (StoC)
Invoke reply_to_HLS_authentication method of Current Association object. The messages may be cryptographically protected.
Response to challenge: f (CtoS)
Phase 2: Message exchange
(Protected) DLMS service .request(s)
(Protected) DLMS service .response(s)
Phase 3: AA release
COSEM-RELEASE.request (Client_SAP, Server_SAP)
COSEM-RELEASE.response (Client_SAP, Server_SAP)
Release contexts

It is important to note that the security of the message exchange phase is independent of the authentication mechanism security. Even in the case where the parties are not authenticated cryptographically protected APDUs can be used to ensure message security.

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.

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

• 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

High Level Security (HLS) authentication

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.
Pass1 and Pass 2 are supported by the COSEM-OPEN service.

Pass1 and Pass 2 are supported by the COSEM-OPEN service.
After Pass2 – provided that the proposed application context and xDLMS context are acceptable – the AA is formally established, but the access of the client is restricted to the method *reply_to_HLS_authentication* of the current "Association SN / LN" object.

Pass3 and Pass4 are supported by the method *reply_to_HLS_authentication* of the "Association SN / LN" LN object(s). If both passes 3 and 4 are successfully executed, then the AA is established.
Otherwise, either the client or the server aborts the AA.

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*.

Security context

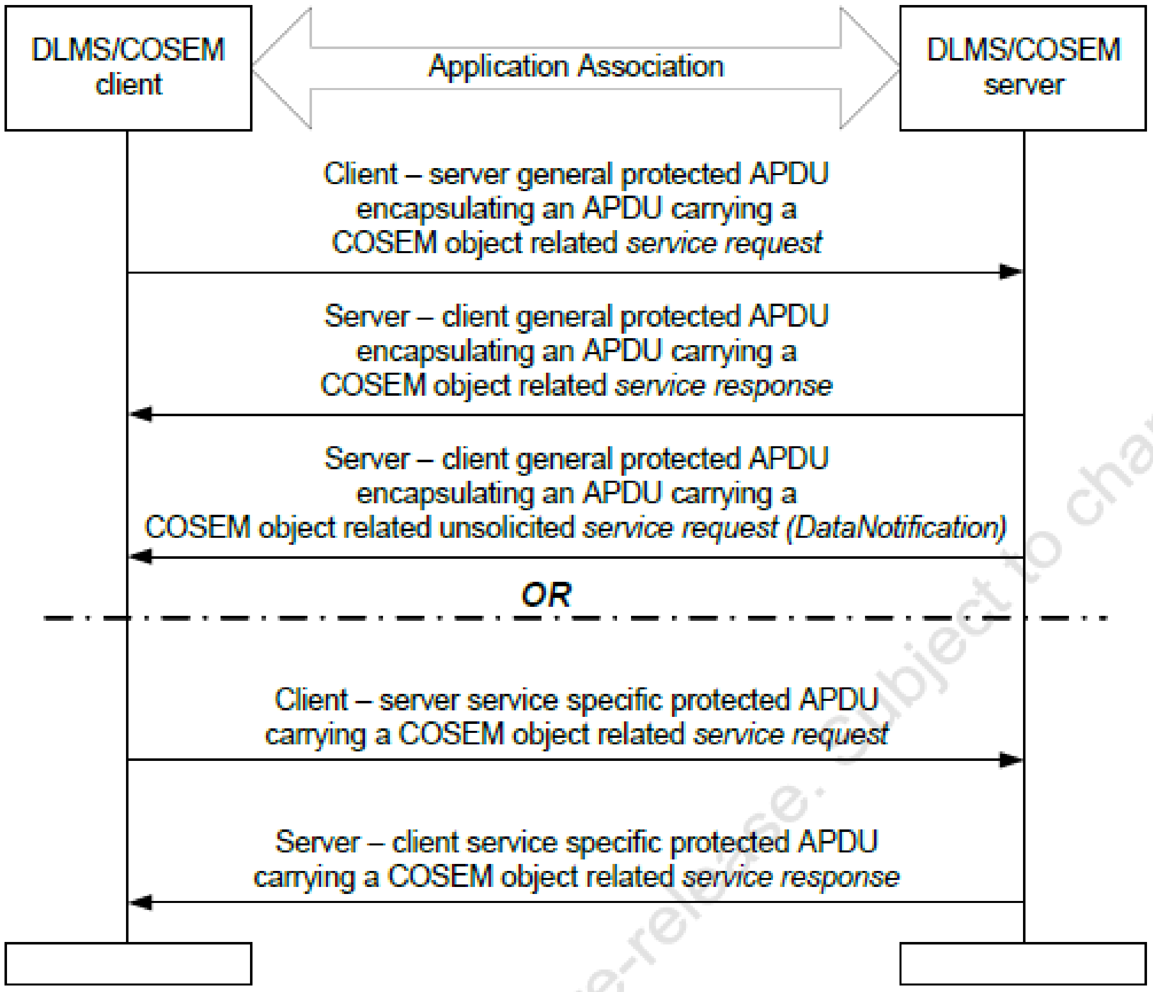
The security context defines security attributes relevant for cryptographic transformations and includes the following elements:
• the security suite, determining the security algorithms available,
• the security policy, determining the kind(s) of protection to be applied generally to all xDLMS APDUs exchanged within an AA.
• 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.

Access rights

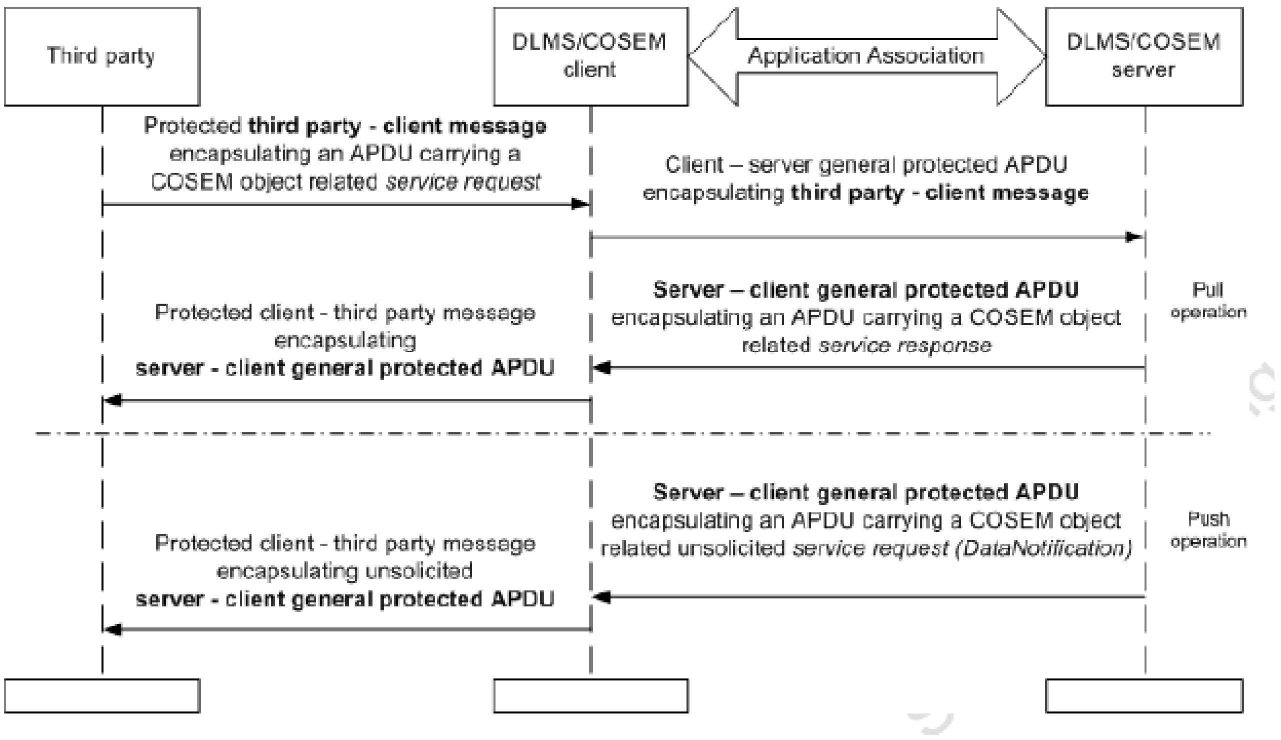
Access rights to attributes may be: *no_access*, *read_only*, *write_only*, or *read_and_write*. Access rights to methods may be *no_access* or *access*

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

Application layer message security


DLMS/COSEM client
Application Association
DLMS/COSEM server
Client – server general protected APDU encapsulating an APDU carrying a COSEM object related *service request*
Server – client general protected APDU encapsulating an APDU carrying a COSEM object related *service response*
Server – client general protected APDU encapsulating an APDU carrying a COSEM object related unsolicited *service request (DataNotification)*
OR
Client – server service specific protected APDU carrying a COSEM object related *service request*
Server – client service specific protected APDU carrying a COSEM object related *service response*

Client – server message security concept


Third party
DLMS/COSEM client
Application Association
DLMS/COSEM server
Protected third party - client message encapsulating an APDU carrying a COSEM object related service request
Client – server general protected APDU encapsulating third party - client message
Server – client general protected APDU encapsulating an APDU carrying a COSEM object related service response
Protected client - third party message encapsulating server - client general protected APDU
Pull operation
Server – client general protected APDU encapsulating an APDU carrying a COSEM object related unsolicited service request (DataNotification)
Protected client - third party message encapsulating unsolicited server - client general protected APDU
Push operation

End-to-end message security concept

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;
• 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.

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.

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

Cryptographic algorithms

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

A cryptographic algorithm and key are used to apply cryptographic protection to data (e.g., encrypt 4582 the data or generate a digital signature) and to remove or check the protection (e.g., decrypt the 4583 encrypted data or verify the digital signature). There are three basic types of approved cryptographic 4584 algorithms: 4585

- cryptographic hash functions
- symmetric algorithms
- asymmetric algorithms

Hash algorithm

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 4606 for the output of a hash function include *hash value* and *message digest*.

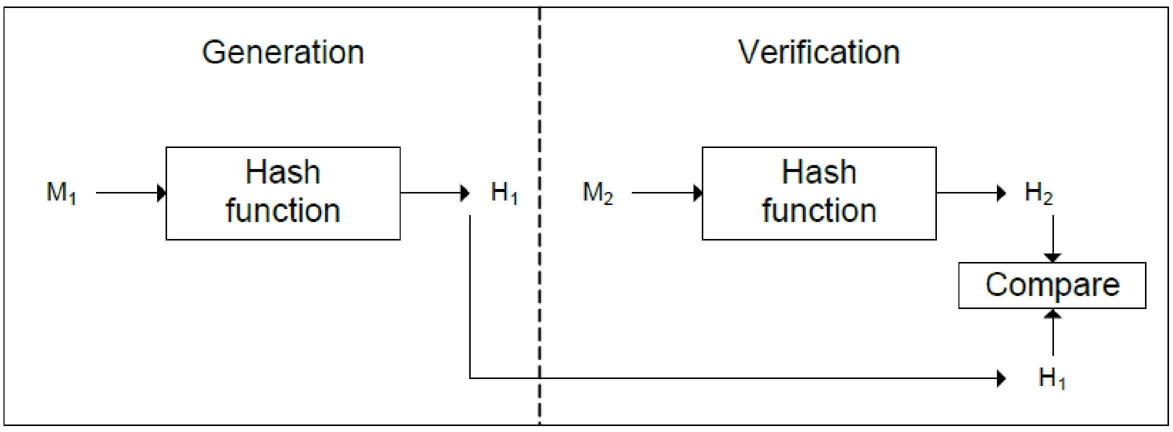


Figure 63 – Hash function

Figure 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).

Hash algorithms are used in DLMS/COSEM for the following purposes:
• digital signature,
• key agreement,
• HLS authentication

Symmetric key algorithms

Symmetric key algorithms are used in DLMS/COSEM for the following purposes:
• authentication of communicating partners,
• authentication and encryption of xDLMS messages,
• authentication and encryption of COSEM data,

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 algorithms are used to provide confidentiality via encryption, or an assurance of authenticity or integrity via authentication, or are used during key establishment.

Encryption and decryption

depicts the encryption and decryption processes. The plaintext (P) and a key (K) are used 4647 by the encryption process to produce the ciphertext (C). To decrypt, the ciphertext (C) and the same 4648 key (K) are used by the decryption process to recover the plaintext (P).

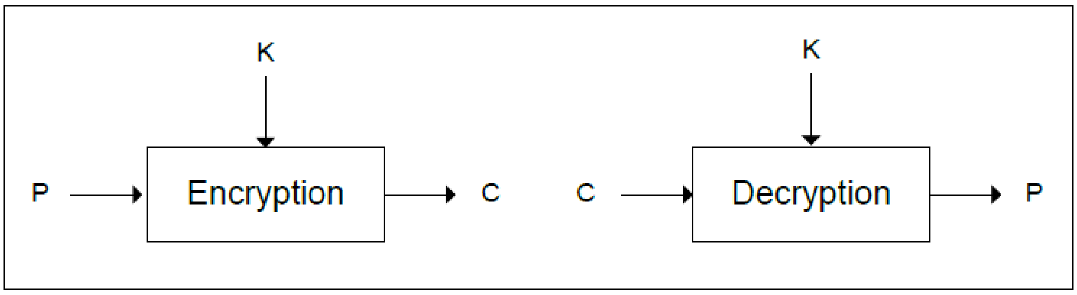


Figure 64 – Encryption and decryption

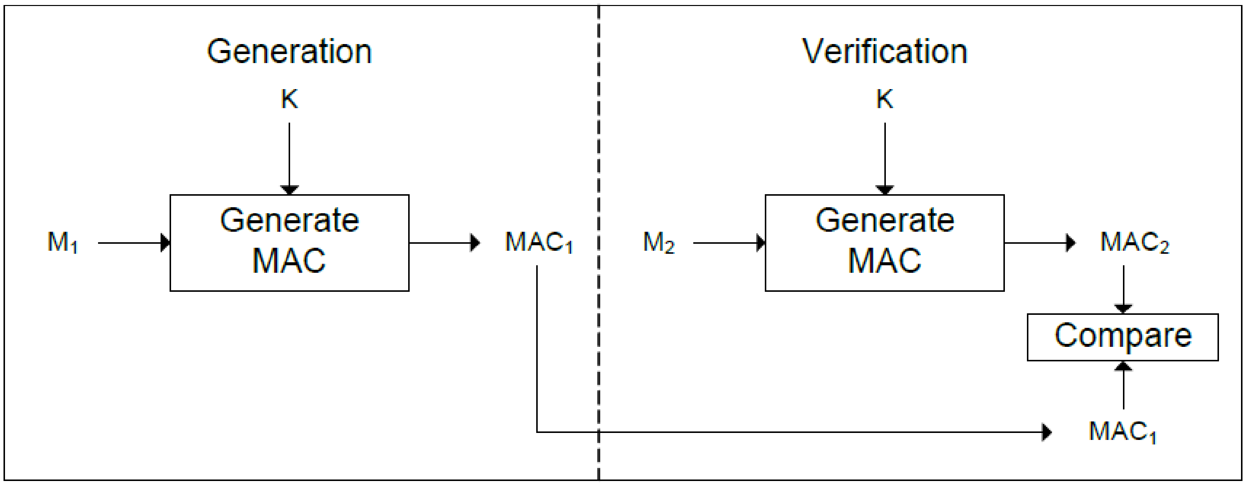
Advanced Encryption Standard

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.

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.

Message Authentication Code

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



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 labeling 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.

Key wrap

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

Galois/Counter Mode

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

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 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

GCM functions

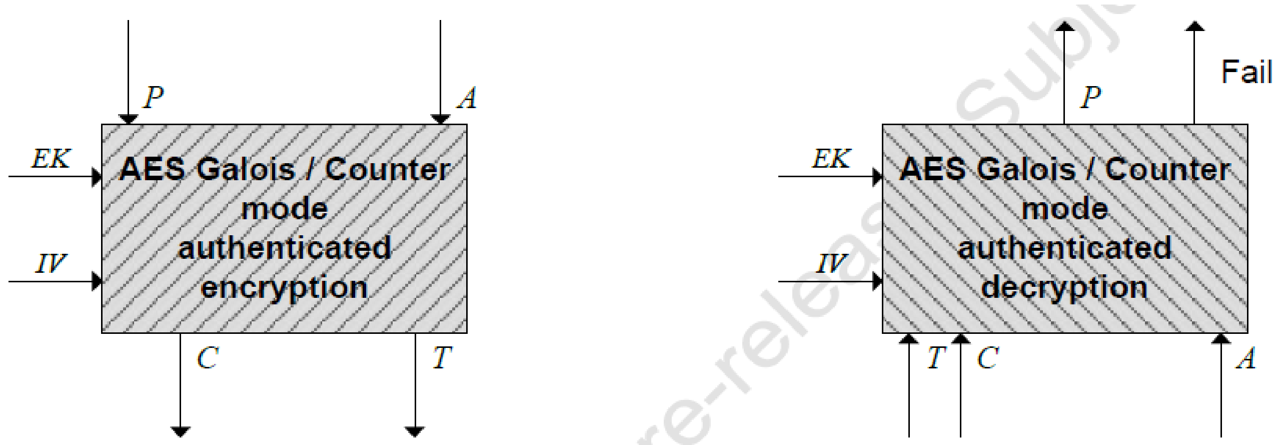

P
A
EK
IV
AES Galois / Counter mode authenticated encryption
C
T
P
Fail
EK
IV
AES Galois / Counter mode authenticated decryption
T
C
A

Figure 66 – 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.

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 bit lengths of the input strings to the authenticated encryption function shall meet the following requirements:
• len(*P*) < 2256;
• 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*.

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*,
• a special error code denoted *FAIL* in this document.

The output *P* indicates that *T* is the correct authentication tag for *IV*, *A*, and *C*; otherwise, the output 4783 is *FAIL*.

The initialization vector, *IV*

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.

The length of the *IV* shall be 96 bits (12 octets): *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
• the trailing (i.e. the rightmost) 32 bits shall hold the invocation field. The invocation field shall be an integer counter.

For each encryption (block cipher) key an invocation counter (*IC*) is maintained separately for the authenticated encryption and the authenticated decryption function. 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 then it is incremented by 1. However, when the maximum value of the *IC* has been reached, any following invocation of the authenticated encryption function shall return an error and the *IC* shall not be incremented

The maximal 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.

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 9.2.3.7 – and shall be:
• for security suite 0 and 1, 128 bits (16 octets): *len(EK)* = 128;
• for security suite 2, 256 bits (32 octets): *len(EK)* = 256;

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.

Length of the authentication tag

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

AES key wrap

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 — and shall be:
for security suite 0 and 1, 128 bits (16 octets): *len(KEK)* = 128;

for security suite 2, 256 bits (32 octets): *len(KEK)* = 256.

Public key algorithms

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

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

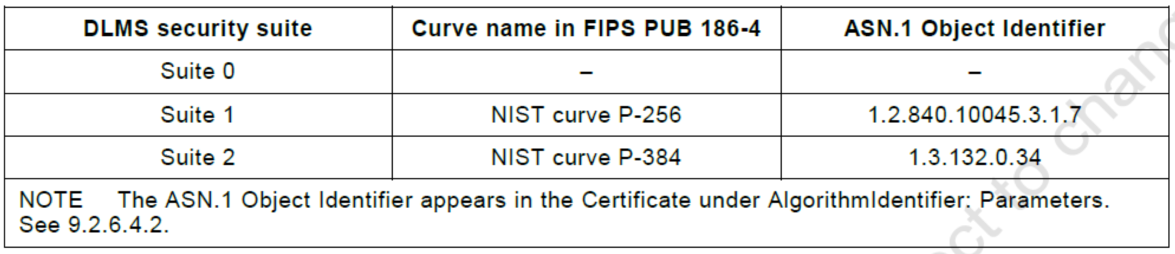
Elliptic curve cryptography
General

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:

[illegible]2= [illegible]3+ [illegible]+ [illegible]

NIST recommended elliptic curves

Table 13 – Elliptic curves in DLMS/COSEM security suites



DLMS security suite	Curve name in FIPS PUB 186-4	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 9.2.6.4.2.		

Data conversions

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 octet strings

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 $b_{l-1}\, b_{l-2}\, ...\, b_0$ of length l shall be converted to an octet string $M_{d-1}\, M_d\, M_0$ 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.

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 $M_{d-1}\, M_{d-2}\, ...\, M_0$ of length d shall be converted to a bit string $b_{l-1}\, b_{l-2}\, ...\, b_0$ of desired length where $d = \lceil l/8 \rceil$ and the leftmost $8d\text{-}l$ bits of the leftmost octet are zero.

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:

The output octet string shall be Md-1 Md-2 … M0.

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 corresponding octet string. A field element $x \in F_p$ is converted to an octet string $M_{d-1}\, M_{d-2}\, ...\, M_0$ of length $d = \lceil log256\ p \rceil$ by applying I2OS conversion primitive with parameter l, where

$$\text{FE2OS}(x) = \text{I2OS}(x,l).$$

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 corresponding field element. An octet string $M_{d-1}\, M_{d-2}\, ...\, M_0$ of length d is converted to field element $x \in F_p$ by applying OS2I conversion primitive where:

• OS2FE(x) = OS2I(x) mod p.

Digital signature

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.

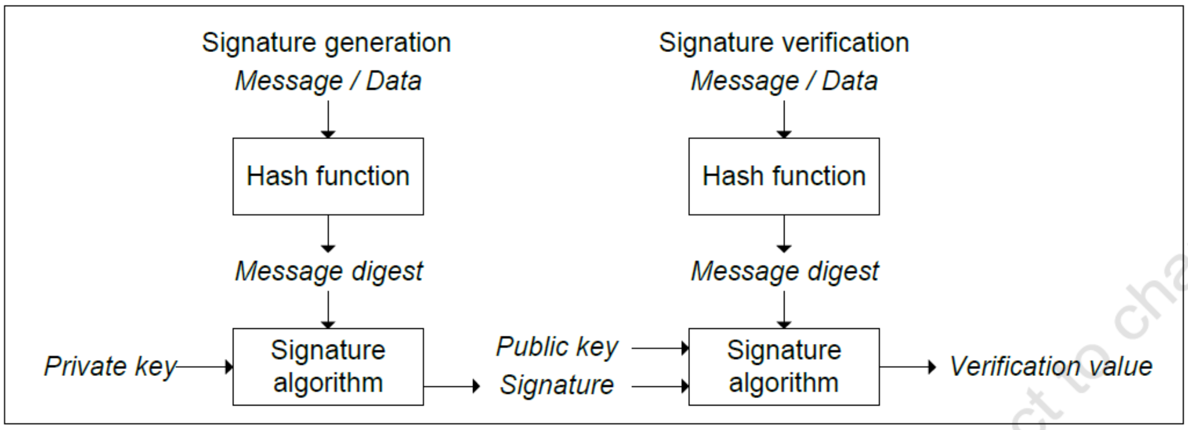


Figure 67 – Digital signatures

Figure 67 depicts the digital signature process. A hash function 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

Elliptic curve digital signature (ECDSA)

For DLMS/COSEM the elliptic curve digital signature (ECDSA) algorithm as specified in FIPS PUB 186-4 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;

• the private key of the signatory, d.

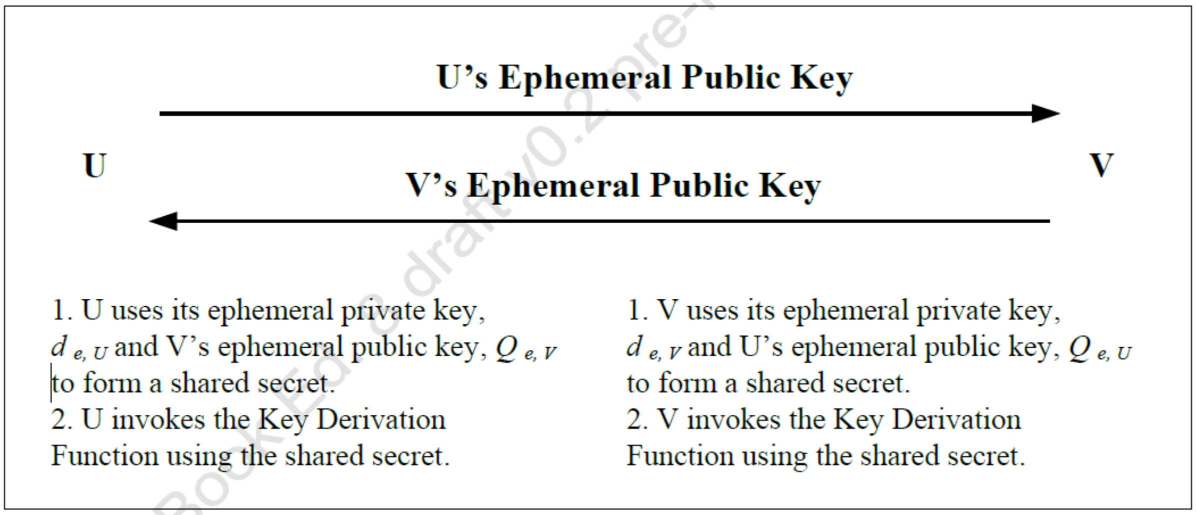
The output is the ECDSA signature (r, s) over M.

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.

Key agreement

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

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


U's Ephemeral Public Key
U
V
V's Ephemeral Public Key
1. U uses its ephemeral private key, $d_{e,U}$ and V's ephemeral public key, $Q_{e,V}$ to form a shared secret.
2. U invokes the Key Derivation Function using the shared secret.
1. V uses its ephemeral private key, $d_{e,V}$ and U's ephemeral public key, $Q_{e,U}$ to form a shared secret.
2. V invokes the Key Derivation Function using the shared secret.

NOTE This figure reproduces NSA2 Figure 1.

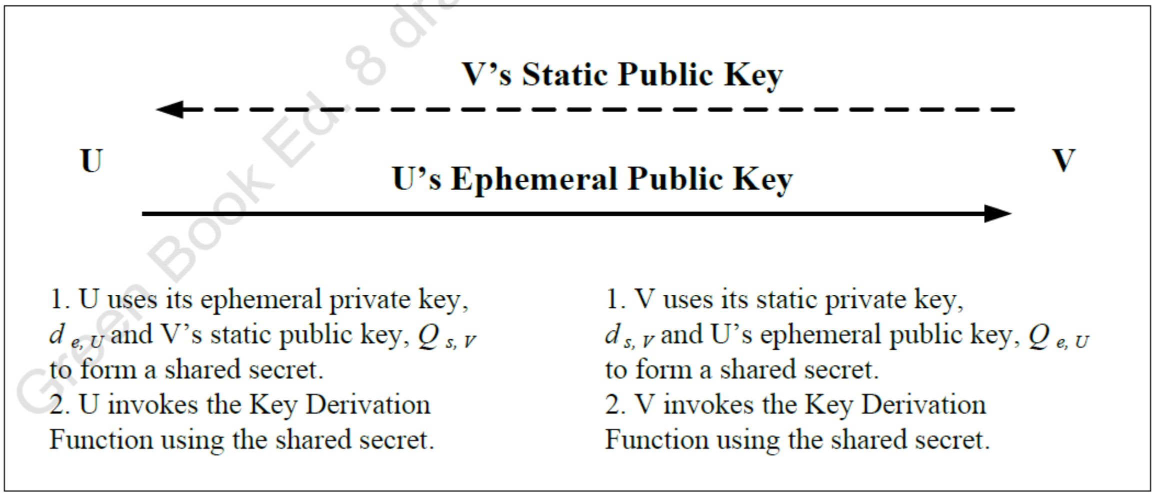
Figure 68 – C(2e, 0s) scheme: each party contributes only an ephemeral key pair

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 A;
2) the parties have agreed on using the NIST Concatenation KDF; see 9.2.3.4.6.5. 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.

Table 14 – 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($Z_{bytestring}$, $OtherInput$) 2. Zeroize Z and $Z_{bytestring}$	1. Compute kdf($Z_{bytestring}$, $OtherInput$) 2. Zeroize Z and $Z_{bytestring}$
NOTE This table is based on NIST SP 800-56A Rev. 2: 2013 Table 18 and **NSA2** Table 1.		

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


V's Static Public Key
U
V
U's Ephemeral Public Key
1. U uses its ephemeral private key, $d_{e,U}$ and V's static public key, $Q_{s,V}$ to form a shared secret.
2. U invokes the Key Derivation Function using the shared secret.
1. V uses its static private key, $d_{s,V}$ and U's ephemeral public key, $Q_{e,U}$ to form a shared secret.
2. V invokes the Key Derivation Function using the shared secret.

NOTE This figure reproduces NSA2 Figure 2.

Figure 69 – C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair

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 A;

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

3) the parties have agreed on using the NIST Concatenation KDF, see 9.2.3.4.6.5. 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;

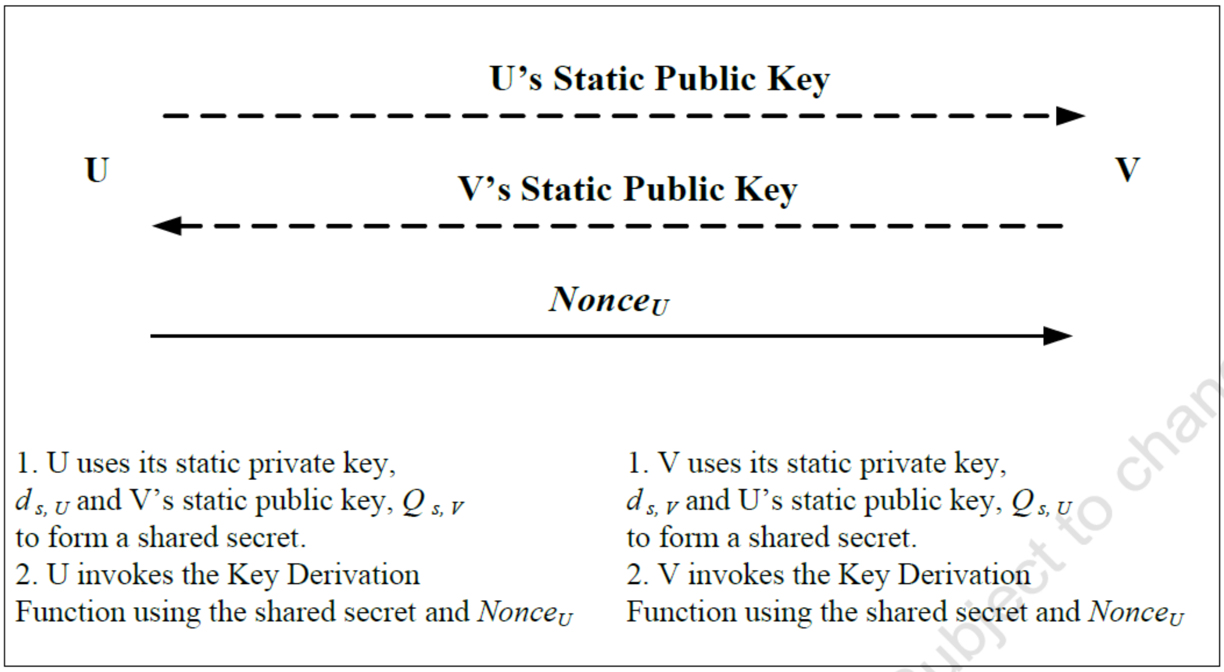
4) prior to or during the key agreement process, the parties obtain the identifier associated with the other party during the key agreement scheme. Party U shall obtain the static public key that is bound to party V's identifier. This static public key shall be obtained in a trusted manner (e.g., from a certificate signed by a trusted CA).

Table 15 – 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($Z_{bytestring}$, $OtherInput$) 2. Zeroize Z and $Z_{bytestring}$	1. Compute kdf($Z_{bytestring}$, $OtherInput$) 2. Zeroize Z, and $Z_{bytestring}$
NOTE This table is based on NIST SP 800-56A Rev. 2: 2013 Table 24 and **NSA2** Table 2.		

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

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



1. U uses its static private key, $d_{s,U}$ and V's static public key, $Q_{s,V}$ to form a shared secret.
2. U invokes the Key Derivation Function using the shared secret and $Nonce_U$

1. V uses its static private key, $d_{s,V}$ and U's static public key, $Q_{s,U}$ to form a shared secret.
2. V invokes the Key Derivation Function using the shared secret and $Nonce_U$

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

Figure 70 – 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 A;

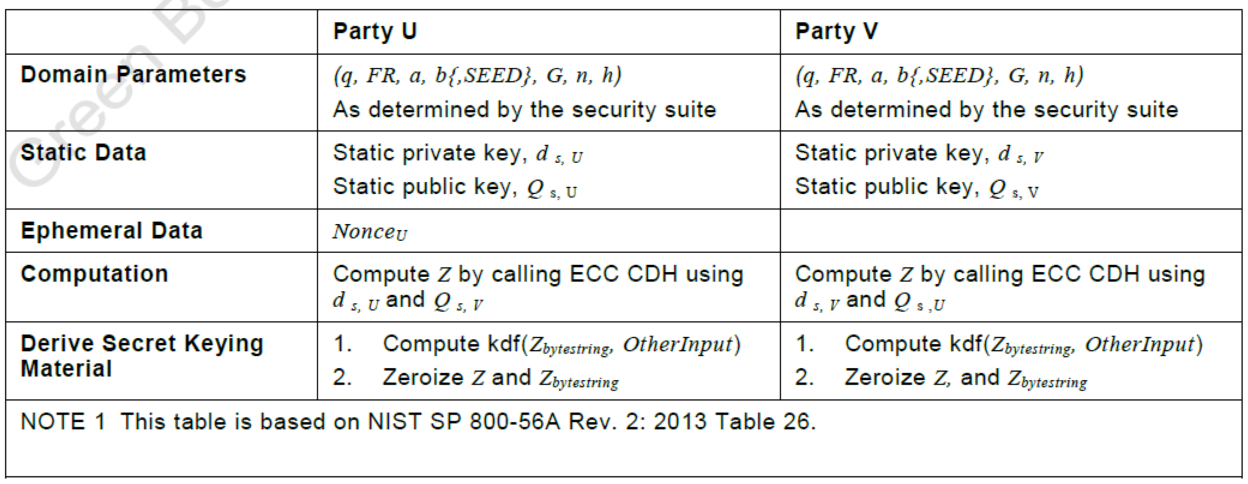
2) each party has been designated as the owner of a static key pair that was generated as specified in using the set of domain parameters, D;

3) the parties have agreed on using the NIST Concatenation KDF, see 9.2.3.4.6.5. 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;

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

5) 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

Table 16 – 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 kdf($Z_{bytestring}$, $OtherInput$) 2. Zeroize Z and $Z_{bytestring}$	1. Compute kdf($Z_{bytestring}$, $OtherInput$) 2. Zeroize Z, and $Z_{bytestring}$
NOTE 1 This table is based on NIST SP 800-56A Rev. 2: 2013 Table 26.		

Key Derivation Function – The NIST Concatenation KDF

Function call: kdf(Z, $OtherInput$)

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

Implementation-Dependent Parameters

Implementation-Dependent Parameters:

6) $hashlen$: an integer that indicates the length (in bits) of the output of the hash function 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.

7) $max\_hash\_inputlen$: an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

NOTE The length shall be less than 2^{64} bits for SHA-256 and less than 2^{128} 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:
8) Z: a byte string that represents the shared secret z.

9) $keydatalen$: an integer that indicates the length (in bits) of the secret keying material to be generated. In DLMS/COSEM it is 128 bit for security suite 1 and 256 bit for security suite 2;
10)$OtherInfo$: 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.

2) *PartyUInfo*: A bit string containing public information that is required by the application 5134 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;

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

.

5) (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.

Table 17 – *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 18
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					

Table 18 – Security algorithm IDs

Algorithm	COSEM cryptographic algorithm ID	Encoded value
AES-GCM-128	2.16.756.5.8.3.0	60 85 74 06 08 03 00
AES-GCM-256	2.16.756.5.8.3.1	60 85 74 06 08 03 01
AES-WRAP-128	2.16.756.5.8.3.2	60 85 74 06 08 03 02
AES-WRAP-256	2.16.756.5.8.3.3	60 85 74 06 08 03 03

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

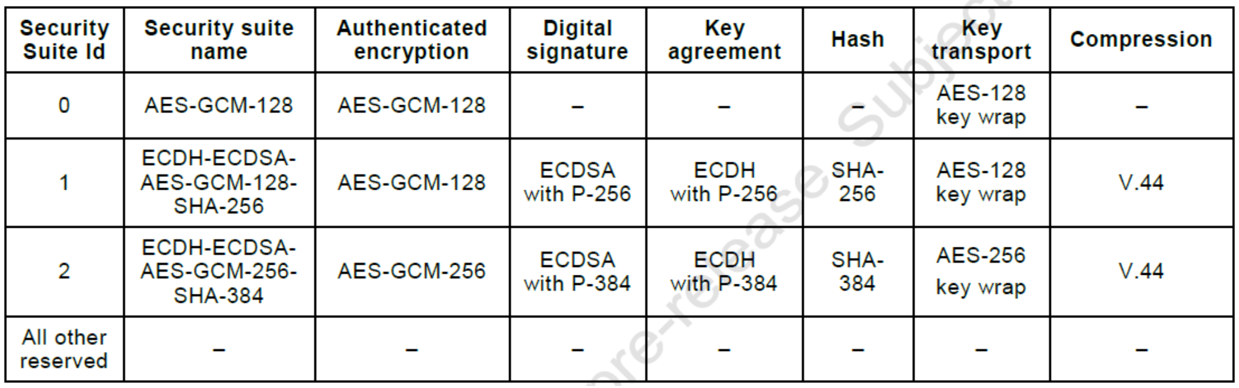
Compression

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 .

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 19 – are based on NSA Suite B and include cryptographic algorithms for authentication, encryption, key agreement, digital signature and hashing specifically

Table 19 – 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	–	–	–	–	–	–	–

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 wrap;
• applying and verifying digital signature;
• key agreement.

9.2.5 Key used with symmetric key algorithms

Symmetric keys types

Symmetric keys are classified according to:

• their purpose:

• a key encrypting key (KEK) is used to encrypt / decrypt other symmetric keys; see 9.2.5.3. In DLMS/COSEM this is the master key;

• an encryption key is used as the block cipher key of the AES-GCM algorithm,

• an authentication key is used as Additional Authenticated Data (AAD) in the AES-GCM algorithm,

their lifetime:

• 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, a broadcast encryption key or an authentication key;

• 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.

• ephemeral keys used generally for a single exchange within an AA.

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,– the authentication key.

33 **Table 20 – 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 Key wrap Key agreement [3]	Can be identified as the KEK in general-ciphering APDUs between client-server, see Table 21
Global unicast encryption key, GUEK [2]	Block cipher key for unicast - xDLMS APDUs and / or - COSEM Data	Key wrap Key agreement [3]	- service specific global ciphering APDU client-server - general-global-ciphering APDU client-server - general-ciphering APDU client-server - "Data protection" object protection parameters
Global broadcast encryption key, GBEK [2]	Block cipher key for broadcast - xDLMS APDUs and / or - COSEM Data	Key wrap Key agreement [3]	
(Global) Authentication key, GAK [2]	Part of AAD to the ciphering process of xDLMS APDUs and / or COSEM data	Key wrap Key agreement [3]	All APDUs between client-server and third party-server
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-dedicated-ciphering APDU client-server during the lifetime of an AA
Ephemeral encryption key	Block cipher key for: - xDLMS APDUs and/or - COSEM data	Key wrap	- 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 37; or

– by the key-id parameter of the general-ciphering APDU, see Table 21;

– or by the protection parameters of the "Data protection " IC, see DLMS UA 1000-1 Ed. 12:2014 4.4.9.

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

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 9.2.3.7.

9.2.5.2 Key information with general-ciphering APDU and data protection

When the general-ciphering APDU is used to protect xDLMS APDUs or when COSEM data are 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

Table 21 – 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 wrap
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 9.2.3.4.6.3; or - the Static Unified Model C(0e,2s, ECC CDH) scheme see 9.2.3.4.6.4.
key-parameters	M	System-Title of the recipient
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 9.2.3.4.6.4 and 9.2.3.4.6.5.
Legend: M: Mandatory (part of a SEQUENCE) S: Selectable (part of a CHOICE) For the ASN.1 specification, see 9.5.		
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.		

9.2.5.3 Key identification

The key identified may be the Global Unicast Encryption Key (GUEK) or the Global Unicast Encryption Key (GUEK). In this case, the key-set bit of the security control byte, see Table 37 – is not relevant and shall be set to zero.

9.2.5.4 Key wrap

Key wrap can be used to establish static or ephemeral symmetric keys.
The algorithm is the AES key wrap algorithm specified in 9.2.3.3.7.7. 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 wrap 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

To establish an ephemeral key using key wrap, 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

9.2.5.5 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 DLMS UA 1000-1

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 9.2.3.4.6.3;

NOTE 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 9.2.3.4.6.4

9.2.6 Keys used with public key algorithms

Table 22 – 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 xDLMS COSEM APDUs and/ or COSEM data or ephemeral public key agreement key. Verifier uses public key to verify digital signature on xDLMS APDUs or COSEM data, or on 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 9.2.3.4.6.2. In the case of the One-Pass Diffie-Hellman C(1e,1s, ECC CDH) scheme only party U has an ephemeral key pair. See 9.2.3.4.6.3
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 9.2.3.4.6.3 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.

9.2.6.3 Public key certificates and infrastructure

9.2.6.3.1 Overview

This clause 9.2.6.3 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 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 , 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 9.2.6.3.2.
- A PKI architecture – as an example – is described in subclauses 9.2.6.3.3.
- The certificate and certificate extension profile is specified in subclauses 9.2.6.4.
- The public key certificates to be held by DLMS/COSEM servers are specified in 9.2.6.5.
- The management of certificates is specified in 9.2.6.6.

9.2.6.3.2 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 5328 PKI example, see 9.2.6.3.3.

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

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

The “Security setup” object 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 Certification Signing Request information on the server to be sent by the client to a CA;
• methods to import, export and remove certificates.

Certificates generally have a validity period. However, certificates issued to DLMS/COSEM servers may be indefinitely valid. Certificates may be renewed 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 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 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.

9.2.6.3.3 PKI architecture – informative

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.

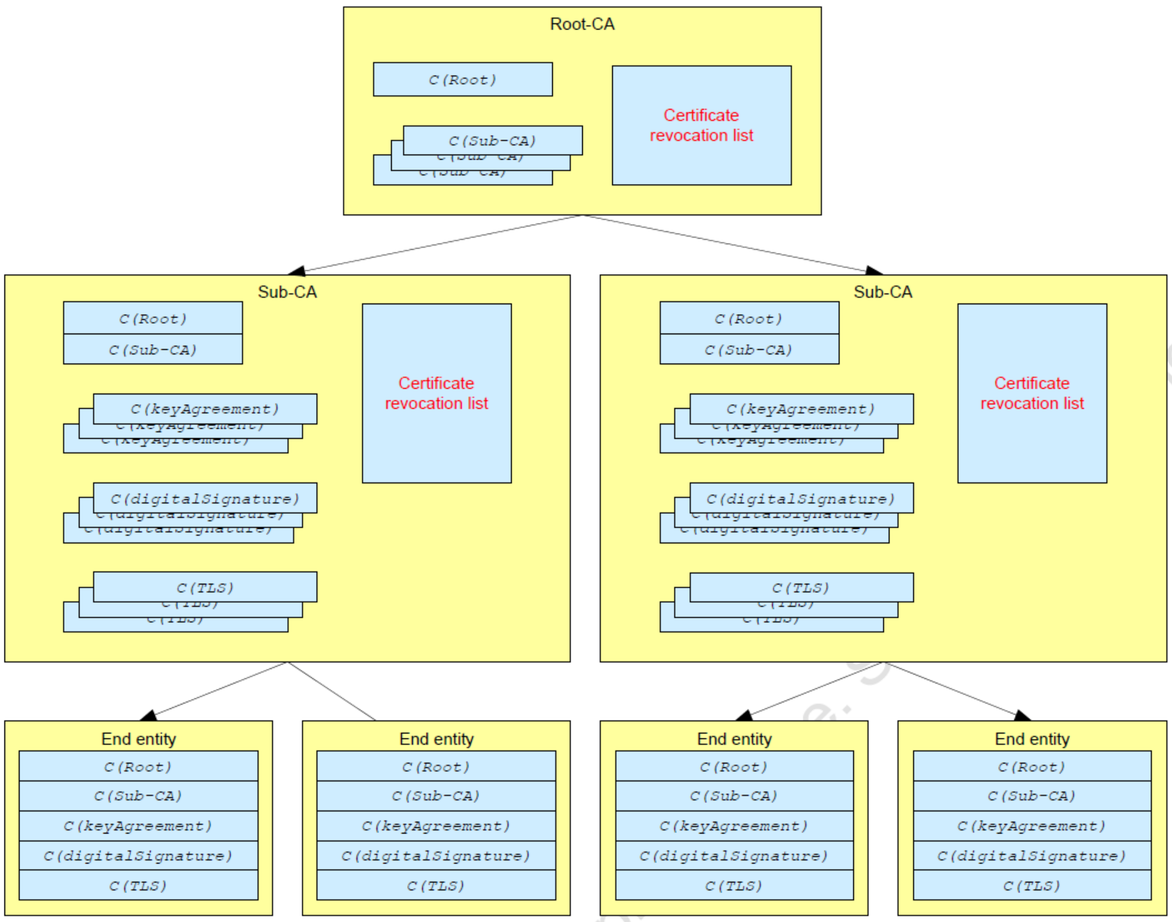


Figure 71 – Architecture of a Public Key Infrastructure (example)

9.2.6.3.3.2 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.

9.2.6.3.3.3 Sub-CA

A Sub-CA is an organisation 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.

9.2.6.3.3.4 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;

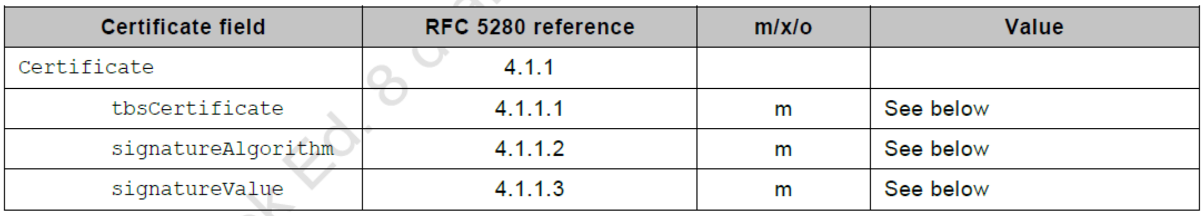
• static key agreement key certificate *`C(keyAgreement)`*, used for key agreement;

• [optional] TLS-Certificate *`C(TLS)`*, used for performing authentication between a DLMS/COSEM client and a DLMS/COSEM server prior the establishment of a secure TLS channel.

9.2.6.4 Certificate and certificate extension profile

9.2.6.4.2 The X.509 v3 Certificate

Table 23 – 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

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.

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

9.2.6.4.3 tbsCertificate

Table 24 – X.509 V.3 tbsCertificate fields

Certificate field	RFC 5280 reference	Clause	m/x/o	Comment
`tbsCertificate`	4.1.2	9.2.6.4.2		
`Version`	4.1.2.1	–	m	'v3' (value is 2)
`Serial Number`	4.1.2.2	9.2.6.4.3.2	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	9.2.6.4.3.3	m	Distinguished name (DN) of the certificate issuer.
`Validity`	4.1.2.5	9.2.6.4.3.4	m	Validity of the certificate.
`Subject`	4.1.2.6	9.2.6.4.3.3	m	Distinguished name (DN) of the certificate subject.
`Subject Public Key Info`	4.1.2.7	9.2.6.4.3.5	m	
`Issuer Unique ID`	4.1.2.8		x	Not applicable
`Subject Unique ID`	4.1.2.8	9.2.6.4.3.6	o	
`Extensions`	4.1.2.9	9.2.6.4.4	m	

9.2.6.4.3.2 Serial number

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.

9.2.6.4.3.3 Issuer and Subject

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.

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.

Table 25 – 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
Serial Number	SERIALNUMBER	o	<SN>	*14-03-10: Check the meaning of serial number at RDN*
NOTE Values within the less-than – greater-than signs "< >" are to be assigned by the PKI or the CA as applicable.				

Table 26 – 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
Serial Number	SERIALNUMBER	o	<SN>	*14-03-10: Check the meaning of serial number at RDN*
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 27 – 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
Serial Number	SERIALNUMBER	o	<SN>	*14-03-10: Check the meaning of serial number at RDN*
Locality	L	x	<Locality>	Locality where the entity is located
State	S	x	<State>	

9.2.6.4.3.4 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.

To indicate that a certificate has no well-defined expiration date, the `notAfter` SHOULD be assigned the GeneralizedTime value of 99991231235959Z.

9.2.6.4.3.5 SubjectPublicKeyInfo

The field `SubjectPublicKeyInfo` shall have the following structure:
SubjectPublicKeyInfo ::= SEQUENCE {
Algorithm AlgorithmIdentifier, 5485
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`;

The algorithm OBJECT IDENTIFIER shall contain the value
• OID value: 1.2.840.10045.2.1;
• OID description: ECDSA and ECDH Public Key.

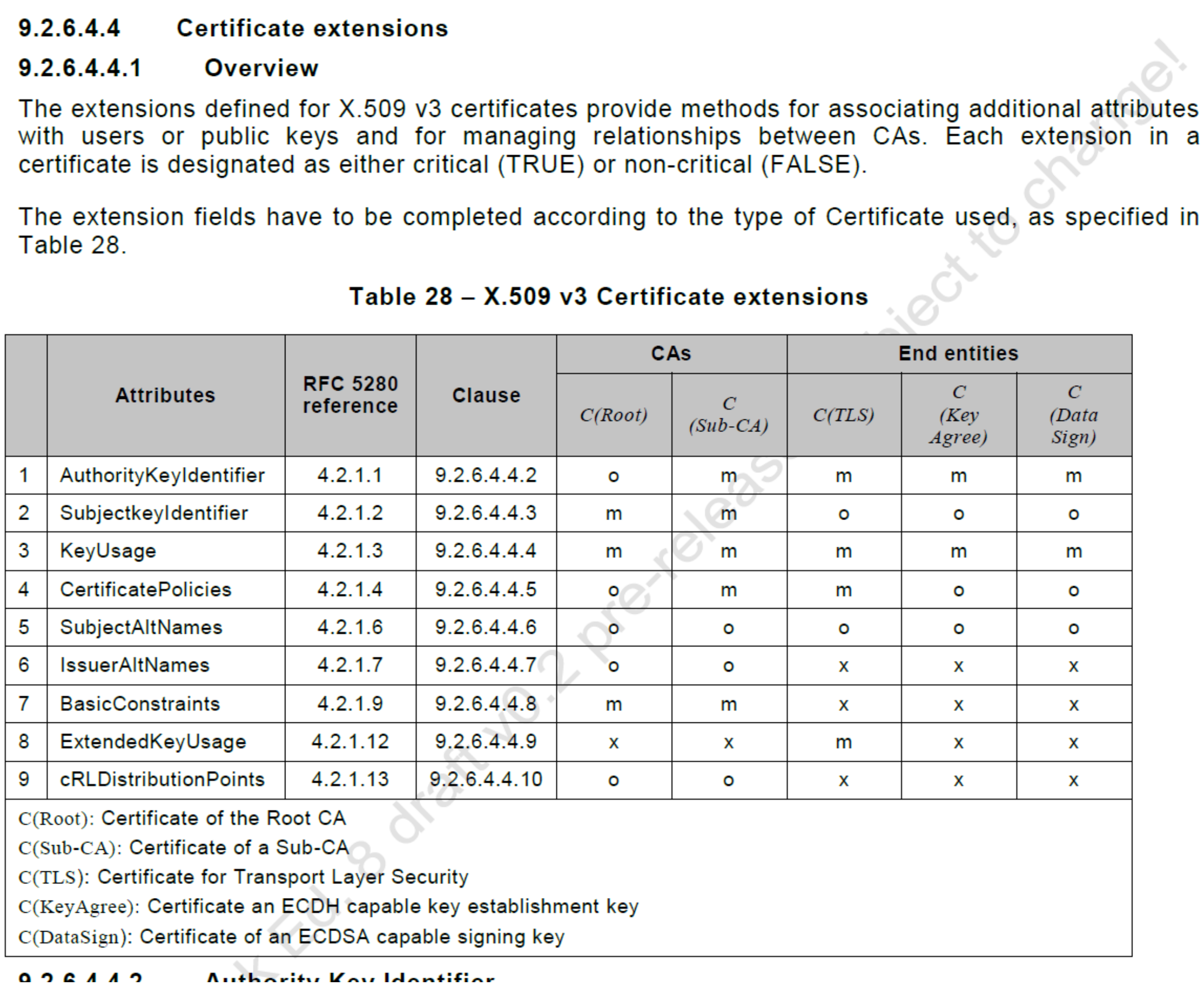
The value of the parameter shall be 1.2.840.10045.3.1.7 for the curve NIST P-256 and 1.3.132.0.34 5498 for the curve NIST P-384.

The subjectPublicKey from SubjectPublicKeyInfo is the ECC public key. ECC public keys have the following syntax:
ECPoint ::= OCTET STRING

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.

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

The public key MUST be rejected if any other value is included in the first octet.



9.2.6.4.4 Certificate extensions

9.2.6.4.4.1 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 28.

Table 28 – 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	9.2.6.4.4.2	o	m	m	m	m
2	SubjectkeyIdentifier	4.2.1.2	9.2.6.4.4.3	m	m	o	o	o
3	KeyUsage	4.2.1.3	9.2.6.4.4.4	m	m	m	m	m
4	CertificatePolicies	4.2.1.4	9.2.6.4.4.5	o	m	m	o	o
5	SubjectAltNames	4.2.1.6	9.2.6.4.4.6	o	o	o	o	o
6	IssuerAltNames	4.2.1.7	9.2.6.4.4.7	o	o	x	x	x
7	BasicConstraints	4.2.1.9	9.2.6.4.4.8	m	m	x	x	x
8	ExtendedKeyUsage	4.2.1.12	9.2.6.4.4.9	x	x	m	x	x
9	cRLDistributionPoints	4.2.1.13	9.2.6.4.4.10	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

9.2.6.4.4.2 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 subclause : 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 subclause : 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).

9.2.6.4.4.3 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;

9.2.6.4.4.4 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 29.

Table 29 – 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

9.2.6.4.4.5 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.
• Value: contains the OID for the applicable certificate policy.

9.2.6.4.4.6 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 thecertificate. 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 ascritical. The `subjectAltName` extension is OPTIONAL otherwise and if included, MUST be marked as critical.

Table 30 – 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.					

9.2.6.4.4.7 IssuerAltName

• Extension-ID (OID): 2.5.29.18;
• Critical: FALSE;
• Description: this extension is used to associate Internet style identities with the certificate issuer

Table 31 – 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.					

9.2.6.4.4.8 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.

Table 32 – 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.					

9.2.6.4.4.9 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

9.2.6.4.4.10 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

9.2.6.5 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 P256 or P384 if the certificate contains a key on P256. The signing CA's key must be P384 if the certificate contains a key on P384.
Depending on the security policy, the following X.509 certificates shall be handled by DLMS/COSEM end entities:

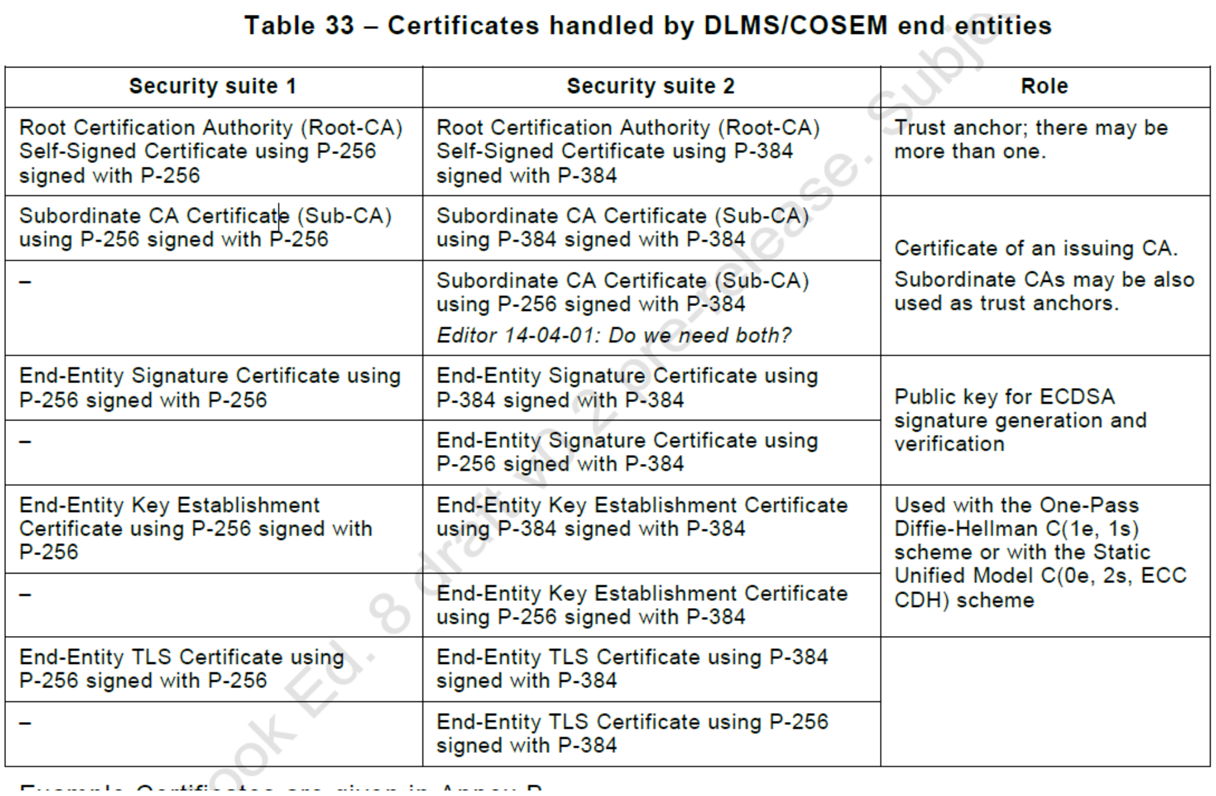


Table 33 – Certificates handled by DLMS/COSEM end entities

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 *Editor 14-04-01: Do we need both?*	
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	

9.2.6.6 Management of certificates

9.2.6.6.1 Overview

This subclause applies only to the management of public key certificates in DLMS/COSEM servers, including:

- provisioning the server with trust anchors,
- provisioning the server with further CA certificates,
- security personalisation of the server,
- provisioning servers with certificates of clients and third parties,
- provisioning clients and third parties with certificates of servers,
- removing certificates

9.2.6.6.2 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.

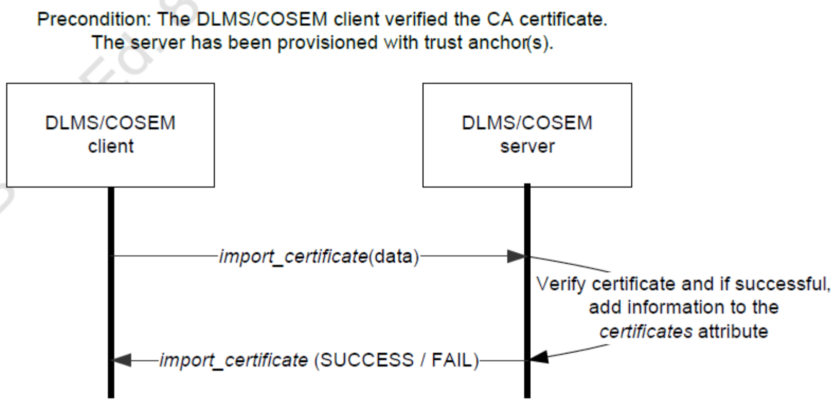
NOTE A server may be provisioned with more than one trust anchor.

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.

9.2.6.6.3 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 72.

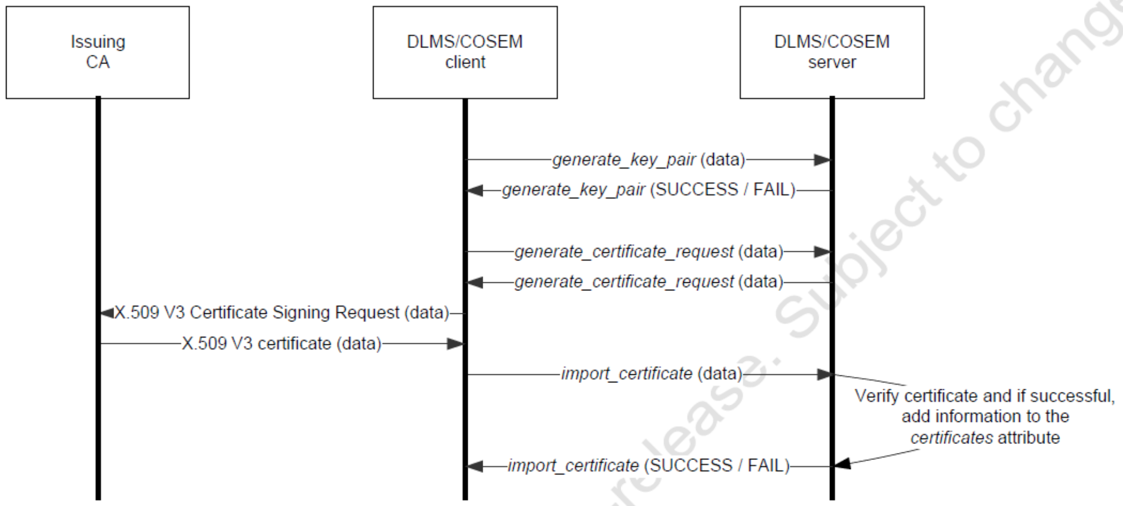

Precondition: The DLMS/COSEM client verified the CA certificate.
The server has been provisioned with trust anchor(s).
DLMS/COSEM client
DLMS/COSEM server
import_certificate(data)
Verify certificate and if successful, add information to the certificates attribute
import_certificate (SUCCESS / FAIL)

Import_CACertificate_GK140402.wmf

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 72 – MSC for provisioning the server with CA certificates

9.2.6.6.4 Security personalisation of the server

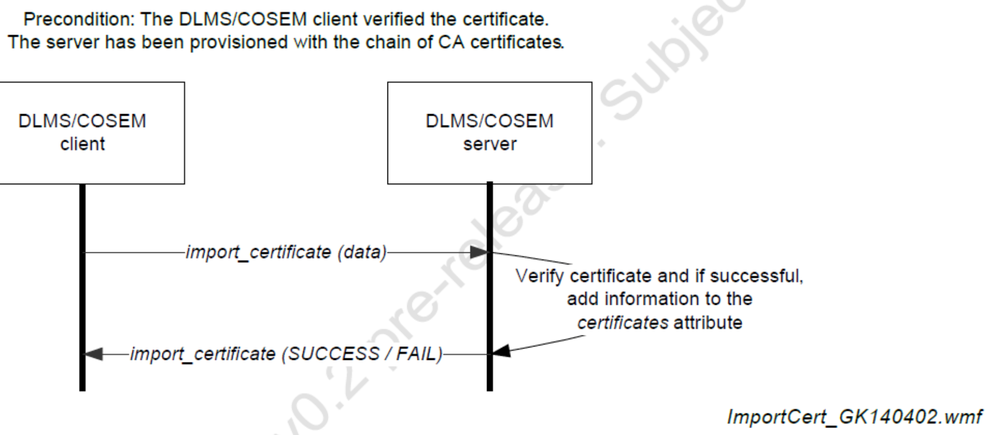

Issuing CA
DLMS/COSEM client
DLMS/COSEM server
generate_key_pair (data)
generate_key_pair (SUCCESS / FAIL)
generate_certificate_request (data)
generate_certificate_request (data)
X.509 V3 Certificate Signing Request (data)
X.509 V3 certificate (data)
import_certificate (data)
Verify certificate and if successful, add information to the certificates attribute
import_certificate (SUCCESS / FAIL)

Server_KeyPairProv_GK140402.wmf

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 73 – MSC for security personalisation of the server

9.2.6.6.5 Provisioning servers with certificates of clients and third parties

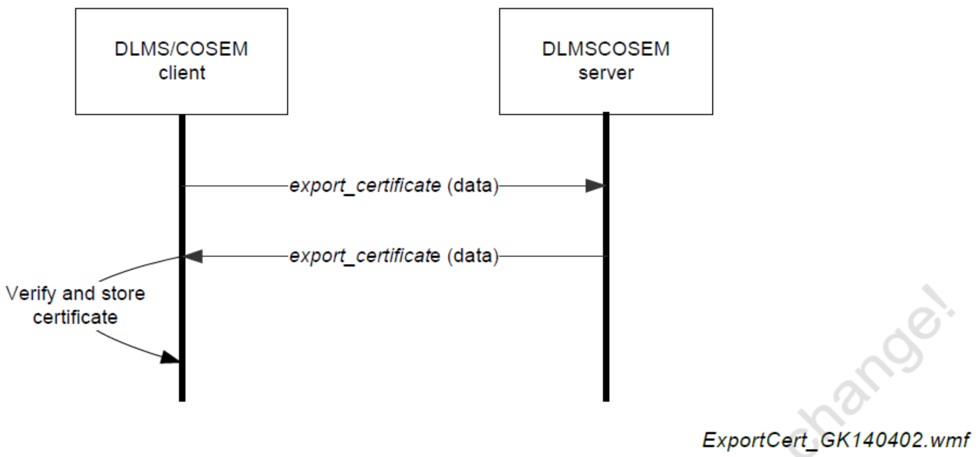

Precondition: The DLMS/COSEM client verified the certificate.
The server has been provisioned with the chain of CA certificates.
DLMS/COSEM client
DLMS/COSEM server
import_certificate (data)
Verify certificate and if successful, add information to the certificates attribute
import_certificate (SUCCESS / FAIL)
ImportCert_GK140402.wmf

NOTE The import_certificate (data) method may be also invoked by a third party, using the client as a broker.

Figure 74 – Provisioning the server with the certificate of the client

9.2.6.6.6 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.


DLMS/COSEM client
DLMSCOSEM server
export_certificate (data)
export_certificate (data)
Verify and store certificate
ExportCert_GK140402.wmf

NOTE The export_certificate (data) method may be also invoked by a third party, using the client as a broker.

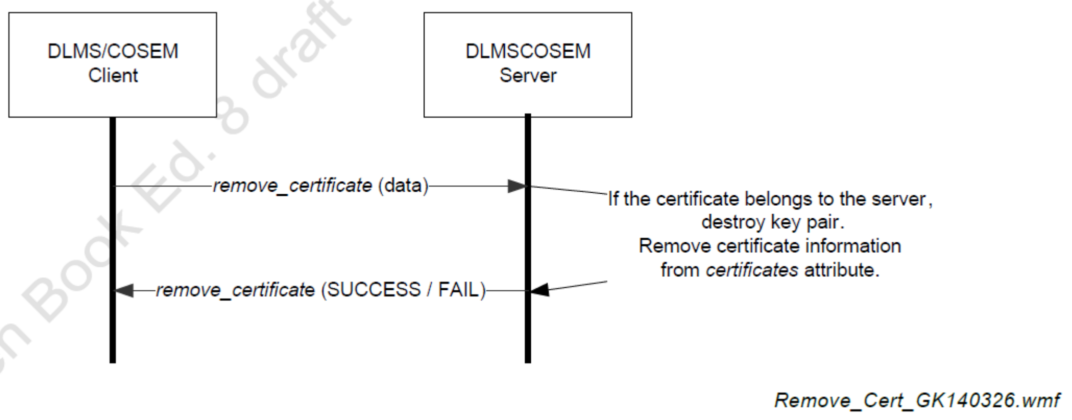
Figure 75 – Provisioning the client / third party with a certificate of the server

9.2.6.6.7 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 document.

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.


DLMS/COSEM Client
DLMSCOSEM Server
remove_certificate (data)
If the certificate belongs to the server, destroy key pair. Remove certificate information from certificates attribute.
remove_certificate (SUCCESS / FAIL)
Remove_Cert_GK140326.wmf

NOTE The remove_certificate (data) method may be also invoked by a third party, using the client as a broker.

Figure 76 – Remove certificate from the server

9.2.7 Applying cryptographic protection

9.2.7.1 Overview

The cryptographic algorithms can be applied:

- to protect the xDLMS APDUs
- to process the challenges during HLS authentication, see 9.2.7.3;
- to protect COSEM data, .

9.2.7.2 Protecting xDLMS APDUs

9.2.7.2.1 Overview

This subclause 9.2.7.2 specifies how the cryptographic algorithms specified in 9.2.3.3 and 9.2.3.4 can be used to protect xDLMS APDUs:

- 9.2.7.2.2 specifies the possible values of security policy and access rights;
- 9.2.7.2.3 presents the types of ciphered APDUs;
- 9.2.7.2.4 specifies the use of symmetric key algorithms for authentication and encryption;
- 9.2.7.2.5 specifies the use of the ECDSA algorithm for digital signature.

9.2.7.2.2 Security policy and access rights values

The security policy is held by the *security_policy* attribute of the “Security setup” IC. It is an *enum* data type. When the enum value is interpreted as an unsigned8, the meaning of each bit is as shown in Table 34.

Table 34 – Security policy values

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 With this, the value (0) means that no cryptographic protection is required, as in version 0 of the “Security setup” IC.	

Access rights are held by the “Association SN / LN” ICs. The access_mode is of data type enum. When the enum value is interpreted as an unsigned8, the meaning of the bits is as shown in Table 35.

Table 35 – Access rights values

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 (4): access with authenticated request enum (253): access with authenticated, encrypted and digitally signed request and response

9.2.7.2.3 Ciphered xDLMS APDUs

The different kind of ciphered xDLMS APDUs are shown in Table 36. See also 9.3.5.

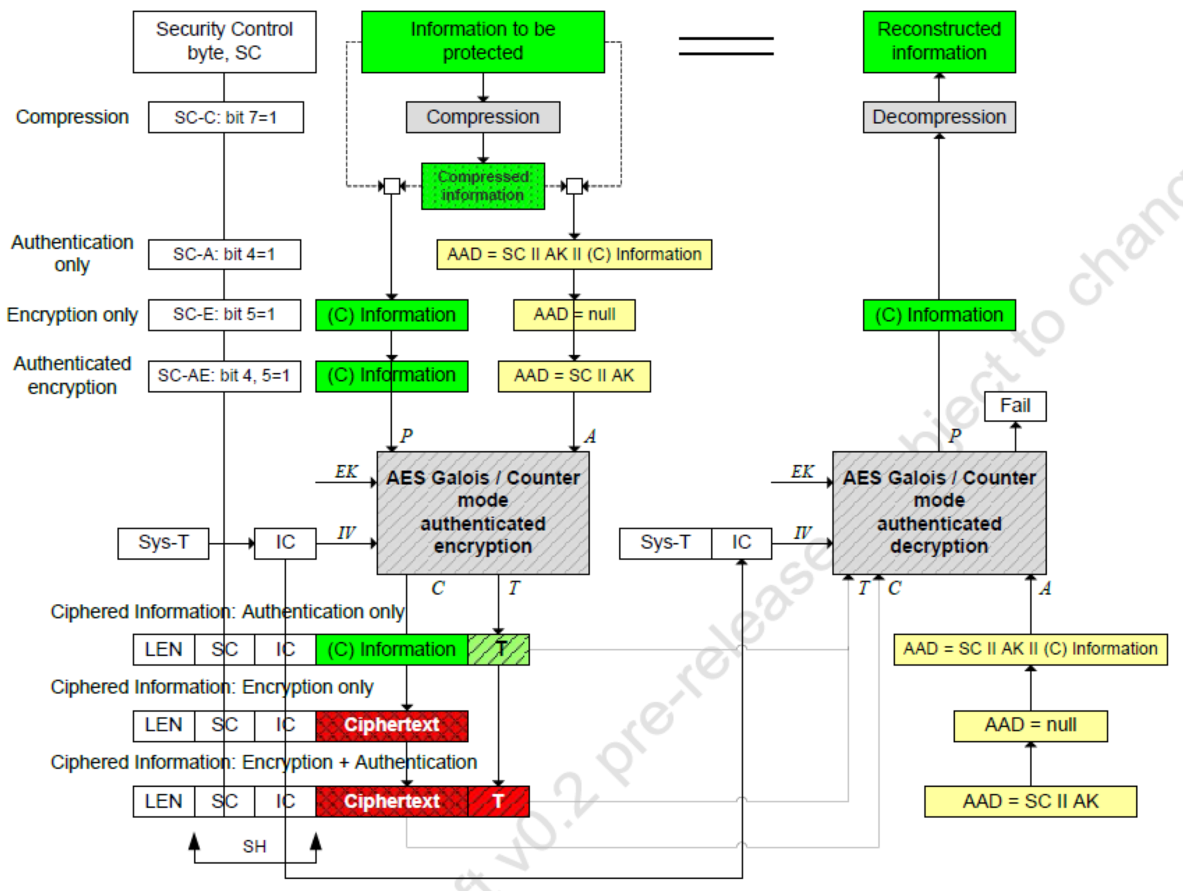
Ciphered xDLMS APDUs can be used in a ciphered application context only. On the other hand, in ciphered application context, both ciphered and unciphered APDUs may be used.

Table 36 – 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, *estab-lished* [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 - Established [2] as part of the exchange [4] Authentication key: global, estab-lished [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 wrap, key agreement or key derivation;
3) In the server, these keys are held by the Security setup objects;
4) Key data are transported in the protected APDU;
5) The use of compression is controlled by the Security Control byte.

9.2.7.2.4 Encryption, authentication and compression


Security Control byte, SC
Information to be protected
Reconstructed information
Compression
SC-C: bit 7=1
Compression
Decompression
Compressed information
Authentication only
SC-A: bit 4=1
AAD = SC II AK II (C) Information
Encryption only
SC-E: bit 5=1
(C) Information
AAD = null
(C) Information
Authenticated encryption
SC-AE: bit 4, 5=1
(C) Information
AAD = SC II AK
Fail
P
A
P
EK
AES Galois / Counter mode authenticated encryption
EK
AES Galois / Counter mode authenticated decryption
Sys-T
IC
IV
Sys-T
IC
IV
C
T
T
C
A
Ciphered Information: Authentication only
LEN
SC
IC
(C) Information
T
AAD = SC II AK II (C) Information
Ciphered Information: Encryption only
LEN
SC
IC
Ciphertext
AAD = null
Ciphered Information: Encryption + Authentication
LEN
SC
IC
Ciphertext
T
AAD = SC II AK
SH

Additional Authenticated Data AAD (Associated data) contains:
- Authentication only: SC II AK II (C) Information;
- Encryption only: Null
- Authenticated encryption: SC II AK
- In the case of the general-ciphering service, it contains additional elements as well.

(C) Information: Compressed information

A = AAD	*P* = Plaintext
AK =Authentication key	SC = Security control byte
C = Ciphertext	SH = SC II IC Security header
EK = Encryption key	*Sys-T* = System title (originator)
IC = Invocation counter	*T* = Authentication tag
IV = Sys-T II IC Initialization vector	

Figure 77 – Cryptographic protection of information using AES-GCM

9.2.7.2.4.2 The security header

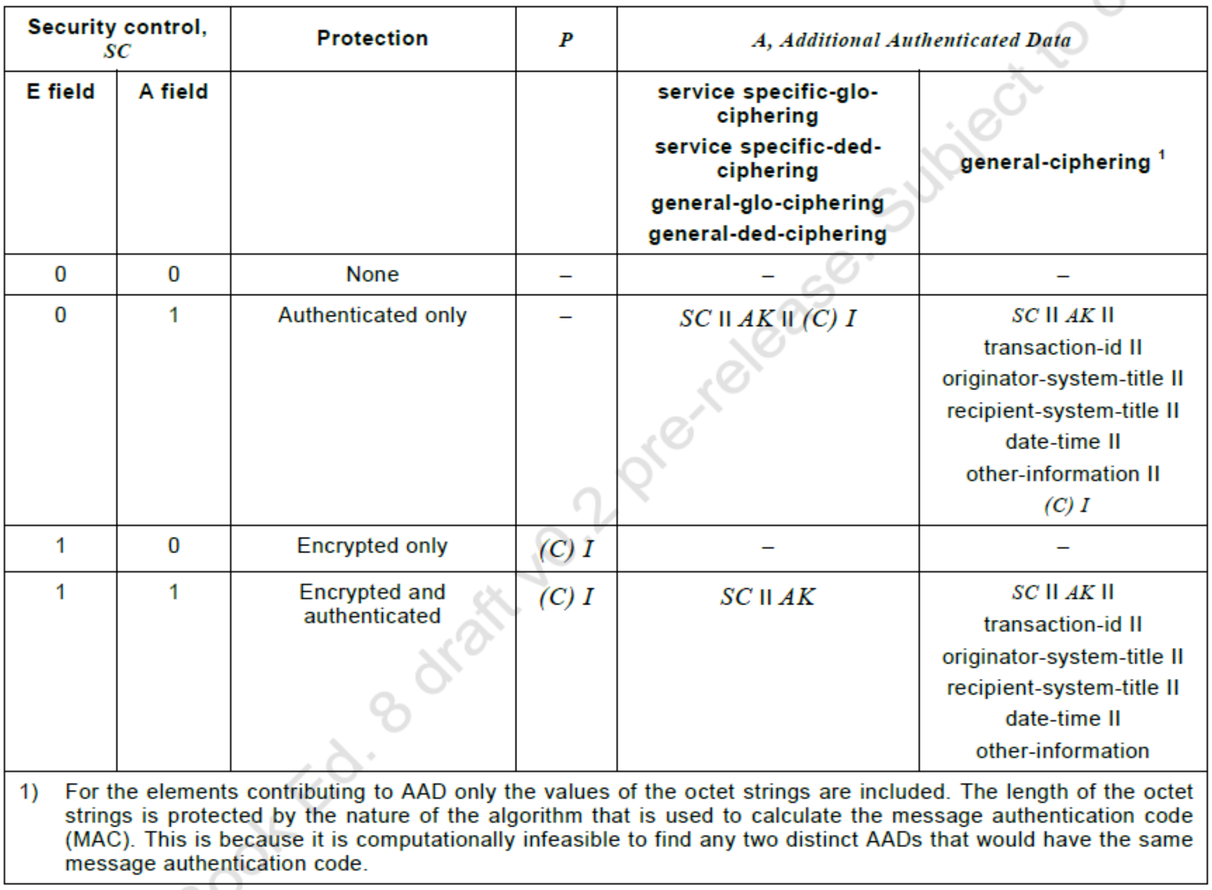
The security header SH includes the security control byte concatenated with the invocation counter: SH = SC II IC

Table 37 – 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-dedicated-ciphering or the general-ciphering APDUs are used.				

9.2.7.2.4.3 Plaintext and additional authenticated data

Table 38 – 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** [1]
0	0	None	–	–	–
0	1	Authenticated only	–	*SC* ‖ *AK* ‖ *(C) I*	*SC* ‖ *AK* ‖ transaction-id ‖ originator-system-title ‖ recipient-system-title ‖ date-time ‖ other-information ‖ *(C) I*
1	0	Encrypted only	*(C) I*	–	–
1	1	Encrypted and authenticated	*(C) I*	*SC* ‖ *AK*	*SC* ‖ *AK* ‖ transaction-id ‖ originator-system-title ‖ recipient-system-title ‖ date-time ‖ other-information

1) For the elements contributing to AAD only the values of the octet strings are included. The length of the octet strings is protected by the nature of the algorithm that is used to calculate the message authentication code (MAC). This is because it is computationally infeasible to find any two distinct AADs that would have the same message authentication code.

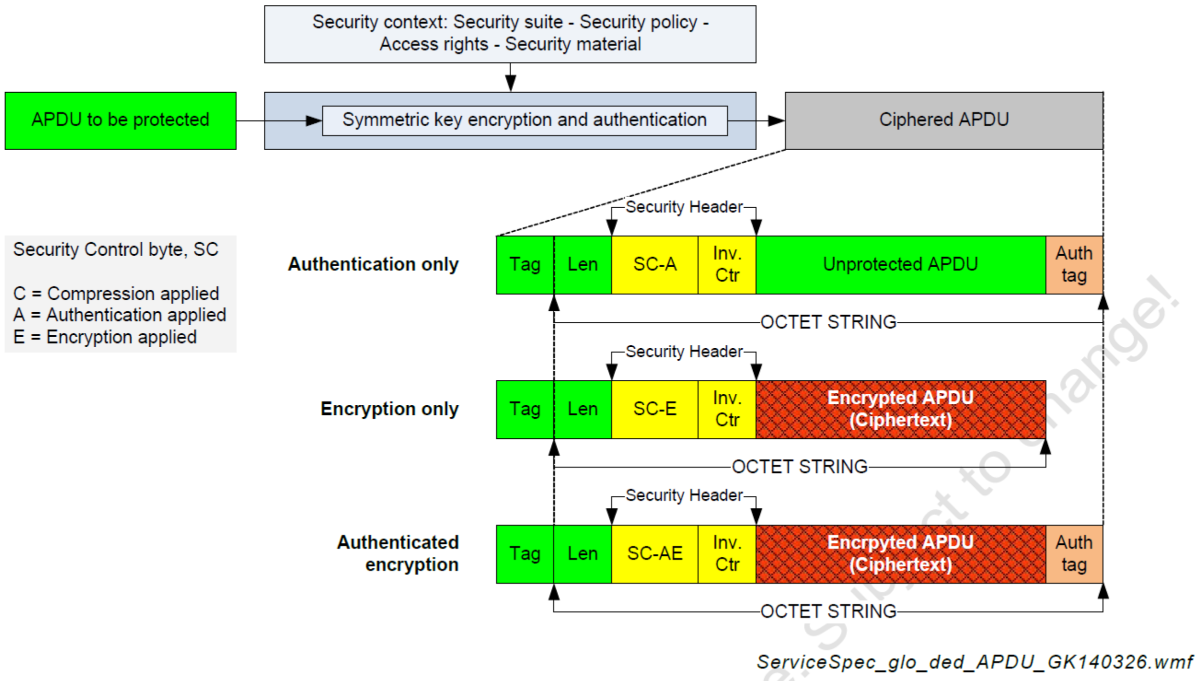
9.2.7.2.4.4 Encryption key and authentication key

These keys used by AES-GCM are specified in 9.2.3.3.7.4 and 9.2.3.3.7.5 respectively

9.2.7.2.4.5 Initialization vector

See 9.2.3.3.7.3.

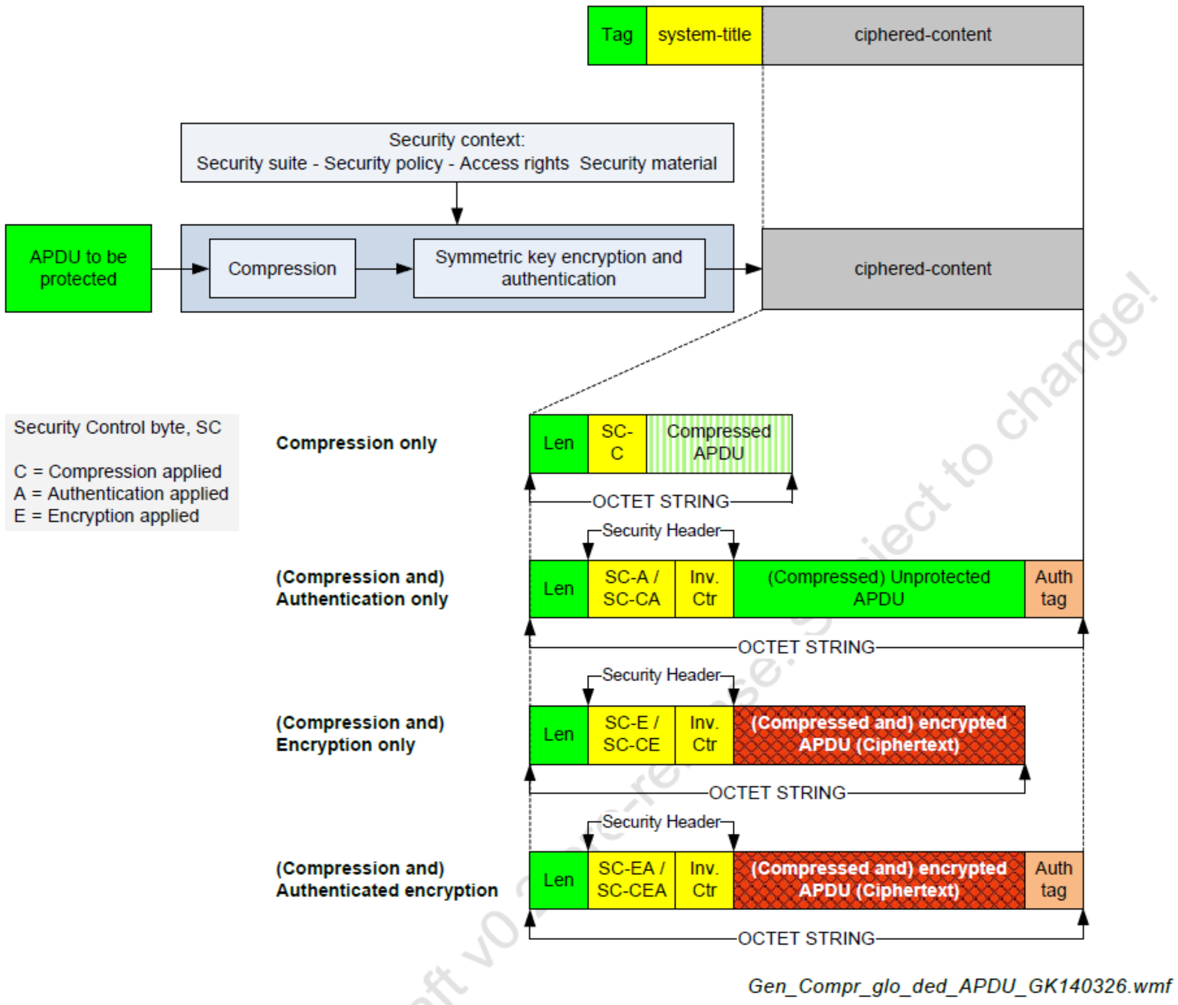
9.2.7.2.4.6 Service-specific ciphered xDLMS APDUs



Security context: Security suite - Security policy - Access rights - Security material
APDU to be protected
Symmetric key encryption and authentication
Ciphered APDU
Security Control byte, SC
C = Compression applied
A = Authentication applied
E = Encryption applied
Authentication only
Encryption only
Authenticated encryption
Security Header
Tag
Len
SC-A
SC-E
SC-AE
Inv. Ctr
Unprotected APDU
Encrypted APDU (Ciphertext)
Auth tag
OCTET STRING
ServiceSpec_glo_ded_APDU_GK140326.wmf

Figure 78 – Structure of service specific global / dedicated ciphering APDUs

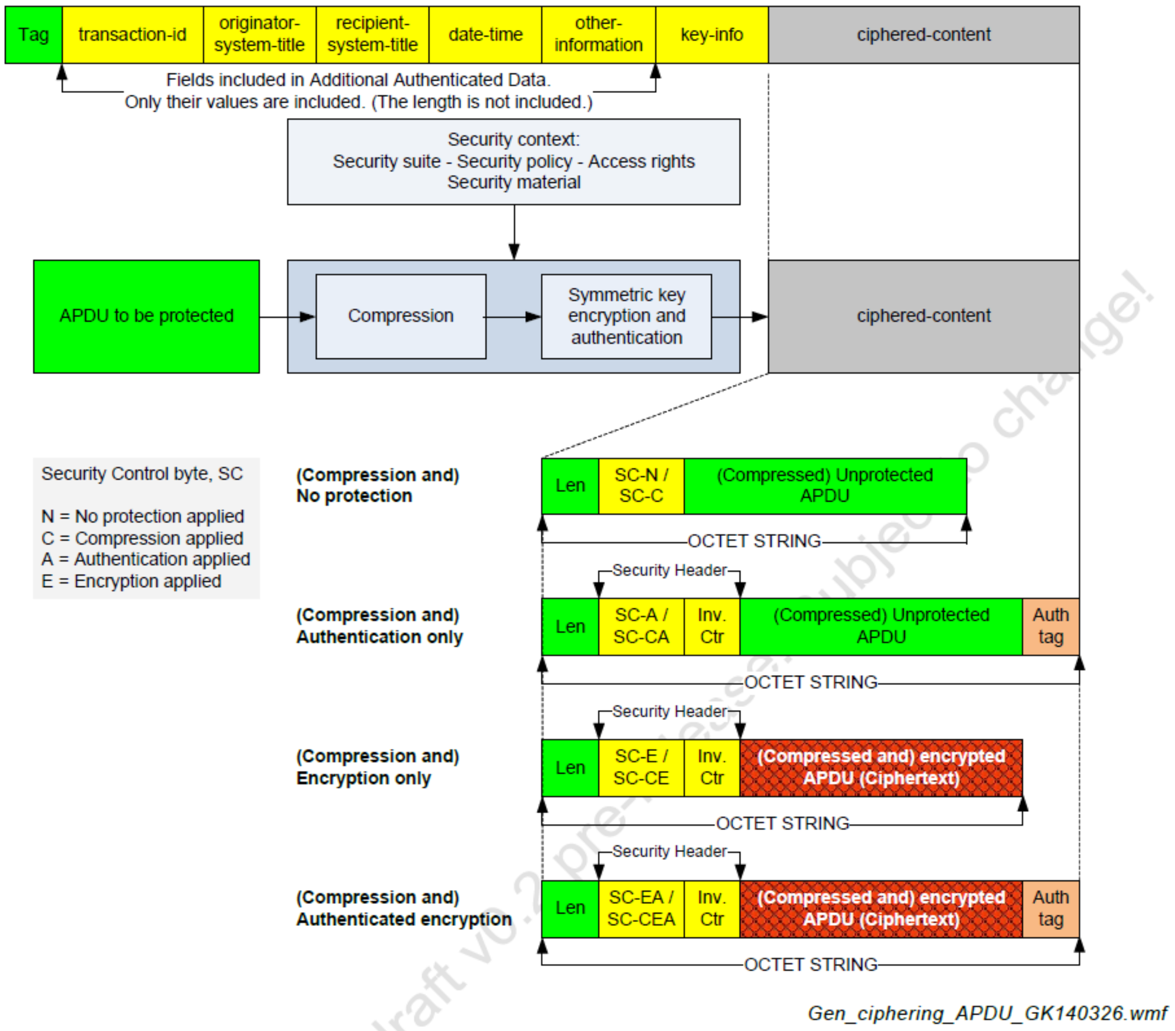
9.2.7.2.4.7 The general-glo-ciphering and general-ded-ciphering xDLMS APDUs



Tag
system-title
ciphered-content
Security context: Security suite - Security policy - Access rights Security material
APDU to be protected
Compression
Symmetric key encryption and authentication
ciphered-content
Security Control byte, SC
C = Compression applied
A = Authentication applied
E = Encryption applied
Compression only
(Compression and) Authentication only
(Compression and) Encryption only
(Compression and) Authenticated encryption
Len
SC-C
Compressed APDU
SC-A / SC-CA
SC-E / SC-CE
SC-EA / SC-CEA
Inv. Ctr
(Compressed) Unprotected APDU
(Compressed and) encrypted APDU (Ciphertext)
Auth tag
Security Header
OCTET STRING
Gen_Compr_glo_ded_APDU_GK140326.wmf

Figure 79 – Structure of general-glo-ciphering and general-ded-ciphering APDUs

9.2.7.2.4.8 The general-ciphering APDU



Tag
transaction-id
originator-system-title
recipient-system-title
date-time
other-information
key-info
ciphered-content
Fields included in Additional Authenticated Data.
Only their values are included. (The length is not included.)
Security context:
Security suite - Security policy - Access rights
Security material
APDU to be protected
Compression
Symmetric key encryption and authentication
ciphered-content
Security Control byte, SC
N = No protection applied
C = Compression applied
A = Authentication applied
E = Encryption applied
(Compression and) No protection
Len
SC-N / SC-C
(Compressed) Unprotected APDU
OCTET STRING
Security Header
(Compression and) Authentication only
Len
SC-A / SC-CA
Inv. Ctr
(Compressed) Unprotected APDU
Auth tag
OCTET STRING
Security Header
(Compression and) Encryption only
Len
SC-E / SC-CE
Inv. Ctr
(Compressed and) encrypted APDU (Ciphertext)
OCTET STRING
Security Header
(Compression and) Authenticated encryption
Len
SC-EA / SC-CEA
Inv. Ctr
(Compressed and) encrypted APDU (Ciphertext)
Auth tag
OCTET STRING
Gen_ciphering_APDU_GK140326.wmf

Figure 80 – Structure of general-ciphering APDUs

Table 39 – Use of the fields of the ciphered 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 ciphered APDU; see 9.5
system-title	–	+	–	See 9.3.5.
transaction-id	–	–	+	
originator-system-title	–	–	+	
recipient-system-title	–	–	+	
date-time	–	–	+	
other-information	–	–	+	
key-info	–	–	+	
security control byte [1]	+	+	+	Provides information on the protection applied, the key-set and the security suite used. See Table 37.
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 9.2.3.3.7.

[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.

9.2.7.2.4.10 Encoding example: global-get-request

Table 40 shows an encoding example using a service-specific global ciphered APDU: a global-get-request APDU

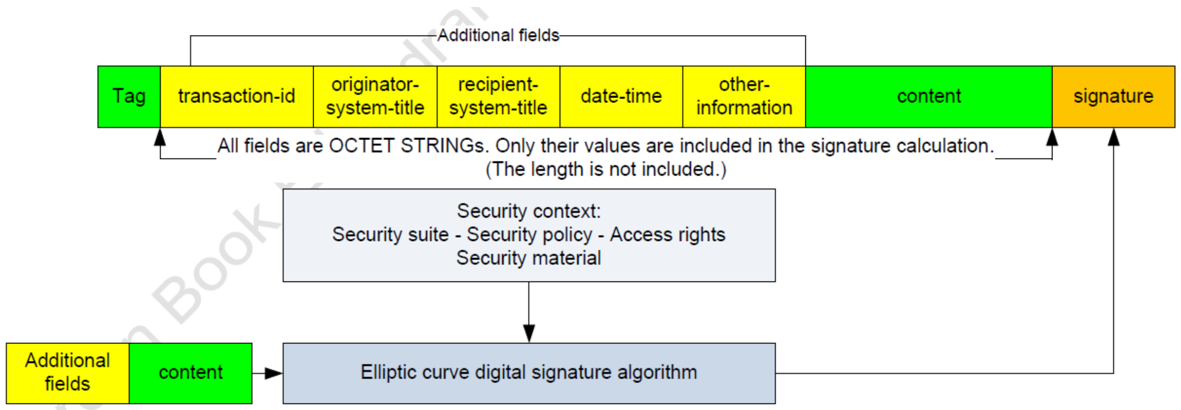
33

Table 40 – global-get-request 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			12	96
		4D4D4D0000BC614E01234567				
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		–	C8122001234567411312FF935A47566827C467BC	–	20	160
	X	Contents			*LEN (X)* bytes	*Len (X)* bits
Authenticated and encrypted APDU		–	–	C81E3001234567411312FF935A47566827C467BC7D825C3BE4A77C3FCC056B6B	32	256

9.2.7.2.5 Digital signature

The algorithm is the elliptic curve digital signature algorithm (ECDSA)


Additional fields
Tag
transaction-id
originator-system-title
recipient-system-title
date-time
other-information
content
signature
All fields are OCTET STRINGs. Only their values are included in the signature calculation. (The length is not included.)
Security context:
Security suite - Security policy - Access rights
Security material
Additional fields
content
Elliptic curve digital signature algorithm

Structure of general-signing APDUs

9.2.7.3 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 general-Ciphering APDU is used. Authenticated encryption is considered to be a single layer of protection;

• to apply digital signature, the general-Signing APDU is used.

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 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.)

9.2.7.4 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 42.

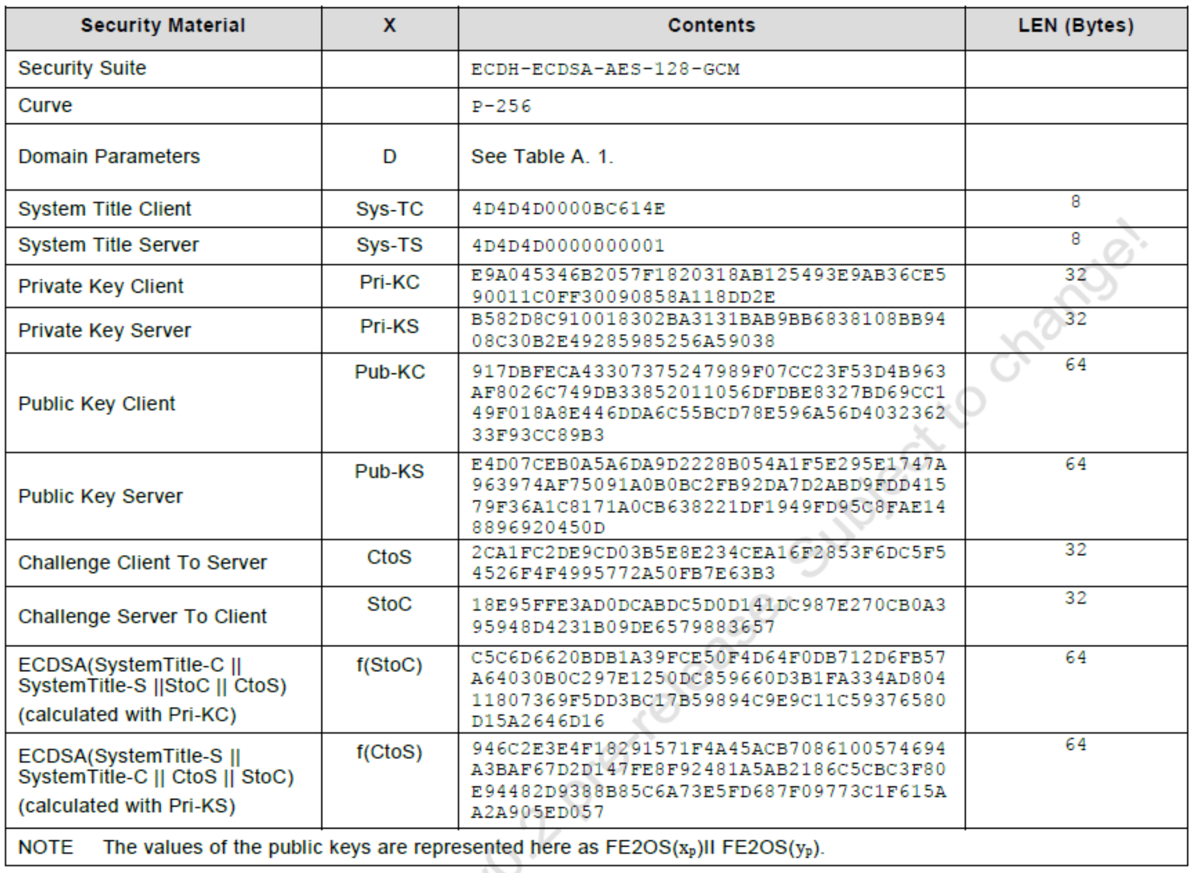
Table 42 – DLMS/COSEM HLS authentication mechanisms

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)
	CtoS: Random string 32 to 64 octets	*StoC*: Random string 32 to 64 octets		

Table 43 – 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		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF5036 77524A323146			
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		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF4B35 366956616759			
T (SC II AK II CtoS)		FE1466AFB3DBCD4F9389E2B7		12	96
f(CtoS) = SC II IC II T		1001234567FE1466AFB3DBCD4F9389E2B7		17	136

Table 44 – HLS example using authentication-mechanism 7 with ECDSA



Security Material	X	Contents	LEN (Bytes)
Security Suite		ECDH-ECDSA-AES-128-GCM	
Curve		P-256	
Domain Parameters	D	See Table A. 1.	
System Title Client	Sys-TC	4D4D4D0000BC614E	8
System Title Server	Sys-TS	4D4D4D0000000001	8
Private Key Client	Pri-KC	E9A045346B2057F1820318AB125493E9AB36CE5 90011C0FF30090858A118DD2E	32
Private Key Server	Pri-KS	B582D8C910018302BA3131BAB9BB6838108BB94 08C30B2E49285985256A59038	32
Public Key Client	Pub-KC	917DBFECA43307375247989F07CC23F53D4B963 AF8026C749DB33852011056DFDBE8327BD69CC1 49F018A8E446DDA6C55BCD78E596A56D4032362 33F93CC89B3	64
Public Key Server	Pub-KS	E4D07CEB0A5A6DA9D2228B054A1F5E295E1747A 963974AF75091A0B0BC2FB92DA7D2ABD9FDD415 79F36A1C8171A0CB638221DF1949FD95C8FAE14 8896920450D	64
Challenge Client To Server	CtoS	2CA1FC2DE9CD03B5E8E234CEA16F2853F6DC5F5 4526F4F4995772A50FB7E63B3	32
Challenge Server To Client	StoC	18E95FFE3AD0DCABDC5D0D141DC987E270CB0A3 95948D4231B09DE6579883657	32
ECDSA(SystemTitle-C || SystemTitle-S ||StoC || CtoS) (calculated with Pri-KC)	f(StoC)	C5C6D6620BDB1A39FCE50F4D64F0DB712D6FB57 A64030B0C297E1250DC859660D3B1FA334AD804 11807369F5DD3BC17B59894C9E9C11C59376580 D15A2646D16	64
ECDSA(SystemTitle-S || SystemTitle-C || CtoS || StoC) (calculated with Pri-KS)	f(CtoS)	946C2E3E4F18291571F4A45ACB7086100574694 A3BAF67D2D147FE8F92481A5AB2186C5CBC3F80 E94482D9388B85C6A73E5FD687F09773C1F615A A2A905ED057	64
NOTE The values of the public keys are represented here as FE2OS(x_p)II FE2OS(y_p).			

9.2.7.5 Protecting COSEM data

The cryptographic algorithms applied to xDLMS APDUs can be also applied to COSEM data. In this case, the list of data to be protected, the required protection and the protection paremeters are determined by the “Data protection” objects;
“Data protection” objects allow:
• specifying the parties applying and removing the protection: the protection may be applied between a client and a server or a third party and a server;

• specifying the required (minimum) protection; this corresponds on the data protection level to the security policy that applies to the protection of xDLMS APDUs;

• specifying the protection parameters that specify the kind of protection: any combination of authentication, encryption and digital signature. The protection parameters can be specified independently for the request and the response of the various services
:
• reading the value of a defined set of attributes with the required protection applied;
• writing the value of a defined set of attributes after removing the protection applied;
• invoking a method with invocation parameters and return parameters.

In addition to the protection of the data, the general security policy prevailing in the given AA and the 5905 specific access rights to the attributes and methods to the “Data protection” object apply.