

Open Metering System

Technical Report 03
XML Key Exchange

Issue 2.0.2 / 2026-11-03

RELEASE

© Open Metering System Group e.V., 2026

Document History

Version	Date	Comment	Editor
1.0.0	2013-10 to 2014-06	Creation of this document	Th. Blank, A. Bolder, U. Pahl, W. Ton, M. Weidinger
1.0.1	2015-01 to 2015-09	Comments due to Voting report on ND 078: vote on OMS_DTR03 – XML Key Exchange 1.0.0 / 2014-06 and resolution of comments	Th. Blank, A. Bolder, U. Pahl, W. Ton, M. Weidinger
1.0.2	2015-10	Issue for release	U. Pahl, A.Bolder
2.0.0	2025-03-14 to 2026-03-17	Revision of whole OMS-TR03 Provide an updated XSD version 3.0 Editorial improvements Release candidate	TF XML Key Exchange A. Reissinger
2.0.1	2026-03-18 to 2026-09-29	Consideration of review comments Release	TF XML Key Exchange A. Reissinger
2.0.2	2026-11-03	Correction in chapter 2.5 Release	Th. Blank, A. Reissinger

Contents

Document History 2

Contents 3

Tables 4

1 Introduction 5
1.1 General 5
1.2 References 5

2 Security 6
2.1 General 6
2.2 Encryption Method 6
2.3 Key Provisioning 6
2.4 Integrity Protection 7
2.5 Formatting and Canonicalization Method 7

3 The Schema File 8
3.1 General 8
3.2 Explanations for Type and Element Names 8
3.3 Optional Vendor Schema 12

4 Examples 13
4.1 Overview 13
4.2 Key Material 13
4.3 Example 1: OMS Basics 14
4.4 Example 2: 3rd Party Network 14
4.5 Example 3: Adapter & Pairing 15
4.6 Example 4: Custom & Vendor 16

Annex A: Applied Tools (informative) 17
A.1 C# implementation 17
A.1.1 General 17
A.1.2 Used Resources 17
A.1.3 XSD-Class 17
A.1.4 Structure 17
A.2 Linux shell 18
A.2.1 General 18
A.2.2 Signing of an XML file 18
A.2.3 Verification of an XML file 18

A.3 PHP 19
A.4 Online tools 19
A.5 Groovy 20
A.6 Python 21

Annex B: Excerpts of OMS-S v5.1 (informative) 22
B.1 General 22
B.2 Excerpt of OMS-S4 22
B.3 Excerpt of OMS-S6 24

Tables

Table 3.1 – Explanations for simple types 8
Table 3.2 – Explanations for complex types 9
Table 3.3 - Explanations for XML elements 10
Table 4.1 - Examples Overview 13
Table 4.2 - Key Material 13
Table B.1 – Predefined OMS-KeyIDs 22
Table B.2 – Key Labels 24

1 Introduction

1.1 General

This document describes how key material of OEDs can be distributed in a secure way between market members. More details about the applied security are provided in chapter 2.

The solution builds upon XML to be cross-platform compatible, and it uses XSD schemata to structure and validate the contents of the files. XSD-files make it possible to enforce a mandatory structure and properties in an XML-file, while at the same time giving vendors the opportunity to fill in extra information. Thus, vendors may add manufacturer specific data (e.g. meter name, production information, order information etc.).

This version of the technical report specifies the version 3.0 of the OMS Key Exchange schema defined in “OMS_KEY_EXCH.xsd”. More details about the schema file are provided in chapter 3.

Several examples shown in chapter 4 illustrate the intended use of the schema file and the resulting XML files. All examples, schemas and tooling’s are provided within a zip-archive alongside with this TR.

1.2 References

References to other documents are marked with square brackets. They are defined in chapter 2 of the Open Metering System Specification Volume 1, v 5.1 ([OMS-S1]). Until [OMS-S1] v 5.1 is not released all references are listed in Annex B.1

2 Security

2.1 General

To ensure confidentiality, any key material contained in the XML file shall be encrypted. This means that each **<Key>** element within the XML file is encrypted with a key encryption key. The integrity and authenticity of the whole file is ensured with help of a digital signature.

2.2 Encryption Method

The encryption process shall use the AES Key Wrap algorithm as specified in RFC 3394. The supported encryption algorithm is:

- AES-256 Key Wrap: http://www.w3.org/2001/04/xmlenc#kw-aes256 (March 2025)

NOTE 1: The key encryption key (wrapping key) used to encrypt **<Key>** elements is therefore 256 bits long. However, it is possible to wrap 128-bit keys as well.

NOTE 2: The CipherValue must be serialised as base64 according to the xmlenc standard.

2.3 Key Provisioning

A 256-bit key encryption key used to encrypt each **<Key>** element shall be provided by one of two methods:

- Key encryption key element (embedded in XML)

  A symmetric key encryption key can be included inside the XML file within a **<KeyEncryptionKey>** element. If this method is used, the key encryption key shall be encrypted using the asymmetric RSA-OAEP encryption algorithm according to:

  http://www.w3.org/2009/xmlenc11#rsa-oaep (March 2025)

  As a prerequisite the public key of the receiver of the file shall be available for the file creator.

- Out-of-Band Key distribution

  Alternatively, the symmetric key encryption key used to encrypt the **<Key>** element can be provided through a separate, secure channel outside the XML file.

When the key encryption key element method is used, the <KeyEncryptionKey> shall carry a <CarriedKeyName> child element that labels the symmetric key contained in the block. Each <KeyData> within the file shall reference that label via a <KeyInfo>/<KeyName> child using the exact same string value; this binding tells the recipient which key encryption key was used to wrap each encrypted <Key>. When the Out-of-Band Key distribution method is used instead, neither <CarriedKeyName> nor the <KeyInfo>/<KeyName> reference is required; the recipient applies the externally agreed key directly.

It is possible to provide several different key encryption keys within an XML file or out-of-band. This is beneficial if different endpoints for the contained keys of the file are intended. For example, a utility can have a sub-contractor that will do network management. In this case the keys for this use can directly be secured with the key encryption key of this sub-contractor whereas other keys are secured with the key encryption key of the utility. If more than one key encryption key is used, they shall be uniquely identified within a <KeyInfo>/<KeyName> element. If the <KeyEncryptionKey> is used, the identification shall be done by using the <CarriedKeyName> element. It is recommended in case of X.509 certification to provide the issuer in a <KeyInfo> element of the respective certificate (see 4.6 for an example).

2.4 Integrity Protection

To ensure that the XML file remains authentic and has not been tampered, a digital signature shall be used. This signature acts as a security seal, allowing the recipient to verify that the content of the file is exactly as it were when originally created.

The digital signature shall cover the entire file, including:

- All device information
- All key material contained within the file

To provide strong security, the signature shall be generated using a cryptographic algorithm such as RSA-SHA256. It will be calculated over the whole file (without the **<Signature>** element itself). The resulting **<Signature>** element will finally be enveloped in the XML root note **<OMSKeyExchange>** inside the file. It is recommended in case of X.509 certification to provide the issuer in a <KeyInfo> element of the respective certificate (see 4.6 for an example).

The public key or the certificate used to verify the digital signature shall be embedded within the **<Signature>** element of the XML. It is crucial that the customer verifies the authenticity of this public key or the certificate, for example, by checking it against a trusted certificate authority or a previously agreed reference. This step ensures that the signature can be trusted and has not been forged.

2.5 Formatting and Canonicalization Method

To ensure interoperable encryption, decryption and signing between users of the file, a certain formatting shall be used. Before signing, the original file needs to be canonicalized to have the same content on several platforms and systems.

So called “pretty printing” can be used for visual aspects before canonicalization and signing.

Exclusive canonicalization shall be used according to this standard:

https://www.w3.org/TR/xml-exc-c14n/

In general parameters shall not be wrapped within one XML tag, e.g. using line breaks.

3 The Schema File

3.1 General

The XML schema OMS_KEY_EXCH.xsd defines the standard structure and format to adhere to comply with this specification. Each XML file that is following this specification shall be validated against this schema. It allows vendor specific enhancements on general level (like order or customer information) and individually for each single device (like product name or number) in the xml file.

The OMS_KEY_EXCH.xsd schema references additional schemas. This is xenc-schema.xsd, which defines the namespace http://www.w3.org/2001/04/xmlenc (March 2025) for the usage of standardized xml encryption mechanisms and xmldsig-core-schema.xsd that provides the namespace http://www.w3.org/2000/09/xmldsig (March 2025) for enabling the standardized xml signature functions.

To enable XML validation without requiring an internet connection, both xenc-schema.xsd and xmldsig-core-schema.xsd must be included alongside the OMS_KEY_EXCH.xsd file. Additionally, the xenc-schema.xsd file from W3C (available at http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd, March 2025) must be modified so that its import statement references a local file instead of an online resource.

All necessary schema files are provided within the attached zip-archive of this technical report.

3.2 Explanations for Type and Element Names

OMS_KEY_EXCH.xsd uses several type definitions and element names. Table 3.1 gives explanations for simple types.

Table 3.1 – Explanations for simple types

Type name	Description
CustomizedAddressType	A simple string type representing an alphanumeric address identification without restrictions. The maximum length is 64 digits. It could be used if other defined address types do not match.
DinAddressType	A simple string type representing the 14-digit DIN address according to DIN 43849, e.g. “3OMG2F12345678”.
EIDType	A simple string type representing the 32-digit numeric eID of an eUICC SIM card.
EUI64Type	A simple string type representing an 8-byte MAC address like “C272181EAC56”. Can be applied e.g. for a LoRaWAN DevEUI.
FourByteType	A simple string type representing a four-byte hexadecimal value, e.g. “3FB83C0A”. Can be applied e.g. for a LoRaWAN DevAddress.
FP2EType	A simple string type representing a 11-digit alphanumeric number according to the definition of the Frensh water association (FP2E). An example is “H25U123456”
HexByteType	A simple string type representing a hexadecimal byte value, e.g. “3F”. Can be applied e.g. for a M-Bus primary address.

Type name	Description
ICCIDType	A simple string type representing the 18-to-20-digit numeric ICCID of a SIM card.
IdentificationNoType	A simple string type representing the 8-digit numeric identification number of an OED according to [EN 13757-7].
IMEIType	A simple string type representing the 15-digit numeric IMEI of a mobile equipment like “352099001761481”.
IMSIType	A simple string type representing the 14-to-15-digit numeric IMSI identification number of a user in a cellular network.
ManufacturerType	ASCII code of the manufacturer ID (three uppercase letters) according to EN 62056-21, e.g. “OMG”. The flag association (https://www.dlms.com/flag-id) administers the allocation of manufacturer IDs.
OMSVersionType	A simple string type representing the implemented OMS version of an OED in the format “xx.yy” whereas x and y can also be just one digit, e.g. “5.1”.
ShortStringType	A simple string type representing a string with the maximum length of 127 characters.
TwoByteType	A simple string type representing a two-byte hexadecimal value, e.g. “3FB8”. Can be applied e.g. for a mioty short address.

Table 3.2 gives explanations for complex types used in the schema file.

Table 3.2 – Explanations for complex types

Element name	Description
DeviceAliasType	A complex type providing optional addressing formats that might be changeable in the field like M-Bus PrimAddr, LoRaWAN DevAddr, mioty ShortAddr, CustomAddr or OwnershipNumber.
DeviceCryptoKeyType	A complex type providing a choice of different key identification types that are each dedicated to specific technologies e.g. like the OMSKey.
DeviceInfoType	A complex type providing additional information that is related to the device like JoinEUI, IMSI, ICCID, EID or OMSVersion.
DeviceType	A complex type representing the device (OED) in the xml schema containing its UniqueID and its CryptoKey elements. It can also contain Alias and Info elements as well as VendorDeviceData.
DeviceUniqueIDType	A complex type representing the unique device identification. It provides several possibilities to achieve this like: MbusAddress, DINAddress, EUI64, FP2E, IMEI or HostRelation.

Element name	Description
HostRelationType	A complex type representing the relation to the “host” and enables the pairing e.g. of a mechanical meter to a radio adapter. The contained DeviceIndex points to the adapter to which this device is connected. The ChildID identifies this device in case the adapter can manage more than one OED.
KeyGenericType	A complex type representing a generic way of key identification that at least requires the definition of a KeyScope. Optionally, it can provide a KeyID and/or a KeyLabel defined by the manufacturer.
KeyIDType	A complex type representing a specific way of key identification that at least requires the KeyID defined according to Table B.1 (in future OMS-S v 5.1: [OMS-S4], Table 5.1). Optionally, it can provide a KeyLabel defined by the manufacturer.
KeyLabelType	A complex type representing a specific way of key identification that at least requires a KeyLabel defined according to Table B.2 (in future OMS-S v 5.1: [OMS-S6], Table C.1). Optionally, it can provide a KeyID defined by the manufacturer.
KeyListType	A complex type representing a list of dedicated keys distinguished with their KeyVersion.
MBusAddressType	A complex type representing the technical M-Bus address.
VendorDataType	A complex type representing a container of manufacturer specific data.

Table 3.3 gives explanations for complex types used in the schema file.

Table 3.3 - Explanations for XML elements

Element name	Description
Alias	An optional element that contains additional technical addresses for the OED.
CryptoKey	This element contains and identifies a key that is assigned to an OED.
CustomAddr	This element contains a customized address of the device.
CustomKey	This is one choice of a CryptoKey element that contains a customized key that is not harmonized by OMS identified by a KeyScope.
Device	The device element is the root element for each contained OED in the OMS key file.
DevAddr	This element contains the 4-byte DevAddr parameter of a LoRaWAN device.
DinAddress	This element contains the cross-manufacturer identification number according to DIN 43849:2024.

Element name	Description
EID	This element contains the 32-digit numeric identifier of an eUICC SIM card.
EUI64	This element contains a 64-bit MAC address format according to IEEE.
FP2E	This element contains a 11-digit alphanumeric address format according to the definition of the Frensh water association (FP2E).
HostRelation	This element contains the relation to the “host” and enables the pairing e.g. of a mechanical meter to a radio adapter (see HostRelationType).
ICCID	This element contains the 18-to-20-digit numeric identification of a SIM card.
IMEI	This element contains a 15-digit numeric address format of a mobile equipment.
IMSI	This element contains the 14-to-15-digit numeric identification number of a user in a cellular network.
Info	An optional element that contains related technical information to the OED.
JoinEUI	This element contains the JoinEUI parameter of a LoRaWAN device.
Key	This element contains the KeyData and the decimal KeyVersion.
KeyData	This element contains all necessary information of the key including its ciphered value.
KeyID	This element identifies a key with a decimal KeyID number. See also KeyIDType definition in Table 3.2.
KeyLabel	This element identifies a key with a string based key name. See also KeyLabelType definition in Table 3.2.
LoRaWANKey	This is one choice of a CryptoKey element that contains a LoRaWAN key identified by a harmonized KeyLabel.
MbusAddress	This element contains the technical M-Bus address elements Manufacturer, IdentificationNo, Version and DeviceType.
miotyKey	This is one choice of a CryptoKey element that contains a mioty key identified by a harmonized KeyLabel.
OMSKey	This is one choice of a CryptoKey element that contains an OMS key identified by a harmonized KeyID.
OMSKeyExchange	This is the root element of the OMS key file. Beside the main content that are the devices with their keys it contains the possibility for VendorOrderData and the TransportKey.
OMSVersion	This element contains the OMS version of an OED.
OwnershipNumber	This element contains the ownership number of an OED.

Element name	Description
PrimAddr	This element contains the one-byte primary address of a wired M-Bus device.
ShortAddr	This element contains the 2-byte ShortAddr parameter of a mioty device.
Signature	The signature element contains all information that is necessary to ensure the integrity and authenticity of the whole xml file.
KeyEncryptionKey	An optional element that contains the ciphered value of the symmetrical key that is used to encrypt the transported device keys.
UniqueID	This element uniquely identifies each contained device (OED).
VendorDeviceData	An optional element that contains device specific data of the manufacturer.
VendorData	An optional element that contains data of the manufacturer or vendor.

3.3 Optional Vendor Schema

An optional vendor-specific XML schema file can be added to provide vendor-defined extensions to the standard structure. This allows vendors to include additional information on a general level (such as custom order or customer data) while maintaining compliance with this specification. Vendor-specific information may be defined and provided either on general level and/or on device level within the XML structure.

If a vendor-specific schema is used, it must be referenced in the XML instance alongside the OMS_KEY_EXCH.xsd. All vendor-specific elements should be defined in a separate namespace to prevent conflicts with the standard schema. Validation of XML files can then be performed against both the standard OMS_KEY_EXCH.xsd and the optional vendor schema to ensure correctness. Usage of a vendor schema is entirely optional; XML files without vendor-specific extensions remain fully compliant using only the standard schema.

4 Examples

4.1 Overview

The four examples show the intended use of the schema for different scenarios. Table 4.1 provides an overview.

Table 4.1 - Examples Overview

Example scenarios	#1 "OMS Basics"	#2 "3rd Party Network"	#3 "Adapter & Pairing"	#4 "Custom & Vendor"
OMSKey	X	X	X	X
LoRaWANKey		X		
miotyKey		X		
OMSKey for Cellular		X		
CustomKey				X
Adapter			X	
Paired meter			X	
Key versioning	X	X	X	
KeyEncryptionKey	Out of band	Included	Out of band	Included
Vendor part				X
Address type	DIN address	EUI64	DIN address	FP2E
Alias type		DevAddress, ShortAddress	Ownership number	
Info type		AppEUI		
Signature style	Public key	Certificate	Public key	Certificate

All xml example files are provided within the zip-archive of this technical report. Also, the necessary certificates and key pairs are provided there. All applied URLs in the examples were valid in March 2025.

NOTE: The provided certificates are self-signed and do not contain a trusted chain.

4.2 Key Material

The following plain text keys are applied in all examples in the order of appearance like shown in Table 4.2. The last key is a 256-bit key, all other keys are 128-bit.

Table 4.2 - Key Material

Plain text keys	Base64 encoded and ciphered keys
0x5708F2152CDF4D8459738B8E5117327A	KEK1: ZMWDML0Mppvd4DzZsMtB7YdIIZ/IVFtQ
0x2F7AA5F47C8B8F2EB6772A6703C84B18	KEK1: 3XfFcKYucsZICQJg3uTqMvB9zzIe/fnV

Plain text keys	Base64 encoded and ciphered keys
0xF82FD6FF3377346C7E23E27C934E6A92	KEK1: WB5PD5DrkLxxCkKExLIoDysx/1yTDN1t
0xE0DED234CBFA57148A9BD13FC307CBFC	KEK1: oQyZaIv3d1V6BxQlro4grARmPlVK74BN
0xB09FD751F4115FEB2A82C099B975B859	KEK1: D646/PINgjNkmNySXm1T8TYoEtZgvNNR KEK2: krhax3qx76U0v8zhEE8LFhDu0fOLtdnj
0x48B45B122634E5F296994AD1F5AACEA1	KEK1: TXcyFJYneTfSKnzKAtmCSuI83Y0AbecJ
0x34A6DB2BDD8D70A9AB31D09ECA087D8E	KEK1: uPI1nGkUGPIKdHZNAgvTjMCYoxmE7R/D
0xFE2B6AF3322418D5183ED3BEA54C458F9 FFC5A484AB1C87C91A7202A9AC7552F	KEK1: vf9mOIlaJd0MJxv0JVrmNxUbnJH2/NRO9P mWo+VCrmnYeEKODZ9jcw==

The following 256-bit key encryption key (KEK1) is applied in all examples:

0xDEADBEEF00123456789ABCCAFEBABE00F31537597B9DBFDE1F304152637485C1

In the examples #2 (see 4.3) and #4 (see 4.6) it is included within the **<KeyEncryptionKey>** element.

Example #4 (see 4.6) additionally applies a second key encryption key (KEK2) in the second **<KeyEncryptionKey>** element. This key is as follows:

0xBEFEEE54044903072669D8FD4B6ED1862A46174D6CB371004C4CE91781E0E583

4.3 Example 1: OMS Basics

The file “OMS_KEY_EXCH_Example_1.xml” is provided within the zip-archive of this technical report.

This example presents an XML-file with mainly mandatory information that complies with the schema file OMS_KEY_EXCH.xsd. The key elements are encrypted with a key encryption key that is supplied out-of-band. The file is digitally signed with the RSA-SHA256 algorithm providing the public key within the file.

It contains two OED’s uniquely identified by their DIN address. The first OED has one key, the Master Key, identified with the KeyID = 0. A KeyLabel element is not necessary as the KeyID clearly defines the usage. The device shows its OMS version 5.1 within the OMSVersion element.

The second OED contains three different key types. The Master Key and two APL security keys (1Fh and 21h; to fulfil the OMS use cases UC-00 and UC-04). Note that the format of KeyIDs in the XML file is decimal, therefore 31 and 33. For the Master Key the file contains two keys, identified with KeyVersion 0 and 1. This device shows its OMS version 5.1 within the OMSVersion element.

4.4 Example 2: 3rd Party Network

The file "OMS_KEY_EXCH_Example_2.xml is provided within the zip-archive of this technical report.

This example presents an XML-file with devices using three different 3rd party communication technologies that complies with the schema OMS_KEY_EXCH.xsd. The key elements are encrypted with a key encryption key that is embedded in the file using RSA-OAEP-MGF1P encryption. The file is digitally signed with an X.509 certificate for authenticity verification.

This file contains the information of 3 devices:

- Device 0 is a water meter that uses OMS over LoRaWAN communication. The device element contains 2 different keys. The KeyIndex = 0 includes the OMS MasterKey with KeyID = 0, which is provided in two different versions (KeyVersion 0 and KeyVersion 1). The KeyIndex = 1 provides the LoRaWAN AppKey that is used for the LoRaWAN OTAA join procedure. The device is identified by its MbusAddress, DINAddress, and EUI64, and includes the JoinEUI for LoRaWAN network joining.
- Device 1 is a thermal energy meter that uses OMS over Cellular communication. The device element contains 2 different keys. The KeyIndex = 0 includes the OMS MasterKey with KeyID = 0. The KeyIndex = 1 provides an OMS COSEKey with KeyID = 96 that is used for additional security features. The device is identified by its MbusAddress, DINAddress, and IMEI, and includes network information such as IMSI, ICCID and EID for cellular connectivity.
- Device 2 is a water meter that uses OMS over mioty for communication. The device element contains 3 different keys. The KeyIndex = 0 includes the OMS MasterKey with KeyID = 0. The KeyIndex = 1 provides the mioty AppKey, and the KeyIndex = 2 provides the mioty NwKey (Network Key). Both mioty keys are required for secure communication in the mioty network. The device is identified by its MbusAddress, DINAddress, and EUI64, and includes a short address alias for efficient addressing in the mioty network.

4.5 Example 3: Adapter & Pairing

The file "OMS_KEY_EXCH_Example_3.xml is provided within the zip-archive of this technical report.

This example presents an XML-file with specially adapter and pairing information that complies with the schema OMS_KEY_EXCH.xsd. The key elements are encrypted with a key encryption key that is supplied out-of-band. The file is digitally signed with the RSA-SHA256 algorithm providing the public key within the file.

This file contains the information of 5 devices:

- Device 0 is an RF-adapter where 2 devices can be connected. As the adapter does also transmit own messages (and therefore represents a Basic OED), the device element contains 3 different keys. The KeyIndex = 0 includes the key that is used by the adapter on its own. The KeyIndex = 1 provide the key that is used by the adapter for a meter that is connected to the child interface 1, the KeyIndex = 2 provide the key that is used by the adapter for a meter that is connected to the child interface 2.
- Device 1 is a meter, that is connected to device 0 as child 1. The data of this meter are encrypted by the adapter (device 0) with the key that is named for ChildId 1.
- Device 2 is a meter, that is connected to device 0 as child 2. The data of this meter are encrypted by the adapter (device 0) with the key that is named for ChildId 2.
- Device 3 is an adapter where only one meter can be connected. As the adapter does also transmit own messages (and therefore represents a Basic OED), the definition of the device includes 2 different keys. The KeyIndex = 0 includes the key that is used by the adapter on its own. The KeyIndex = 1 provide the key that is used by the adapter for a meter that is connected to the child interface 1. Because the meter that will be connected to this adapter is not known, the key exchange file does not include any additional device that is related to this device as its host.

- Device 4 is an adapter where only one meter can be connected. Because the adapter does not transmit own messages, the definition of the device includes only one key. The KeyIndex = 0 includes the key that is used by the adapter for a meter that is connected to the child interface 1. Because the meter that will be connected to this adapter is not known, the key exchange file does not include any additional device that is related to this device as its host.

4.6 Example 4: Custom & Vendor

The file “OMS_KEY_EXCH_Example_4.xml” is provided within the zip-archive of this technical report.

This example presents an XML-file with custom key and additional vendor information that complies with the schema files OMS_KEY_EXCH.xsd and MyVendorNameSpace.xsd. The key elements are encrypted with two different key encryption keys that are embedded in the file using RSA-OAEP-MGF1P encryption. The link to the cipher value of the transported keys is provided with the <CarriedKeyName>. The file is digitally signed with an X.509 certificate for authenticity verification.

This file contains the following information:

- Custom VendorData
- Custom VendorDeviceData
- Device 1 to 4 are a water meters that uses the FP2E notation as unique identifier. Each device has one custom key and contains additional vendor information. Encrypted with the first KeyEncryptionKey (KEK1)
- Device 5 is a wMbus water meter uniquely identified by the DIN address. The device has one custom key and contains additional vendor information. Encrypted with an additional KeyEncryptionKey (KEK2).

NOTE: For the sake of simplicity, the same certificate was used in this example for encrypting KEK1 and KEK2. This must be adjusted in a production environment.

Annex A: Applied Tools (informative)

A.1 C# implementation

A.1.1 General

The example 3 is generated with C# code located in the subfolder “Source” of Example 3. The Source code is generated with Visual Studio 2022.

A.1.2 Used Resources

To build and debug this example source code, the SDK for .NET5.0 or higher is needed. The example code was built and debugged with .NET9.0.

Additionally, to the SDK, the following NuGet Packages are used to realize the cryptographic requirements:

- RFC3394 by Jay Miller in Version 1.0.0
- System.Security.Cryptography.Xml by Microsoft V9.0.9

A.1.3 XSD-Class

To generate the C# class out of the XSD schema files, the XSD-Tool coming within Visual Studio is used. Calling the XSD-Tool inside the “Developer-PowerShell” of Visual Studio, a class file including classes for all XSD Elements is generated. Simply execute the command

```
xsd OMS_KEY_EXCH.xsd xenc-schema.xsd xmldsig-core-schema.xsd /classes
```

inside the “xsd” folder of the example source code from the “Developer-PowerShell”.

Additional information regarding converting XSD-Files into C’ classes can be found with the following link:

https://learn.microsoft.com/de-de/dotnet/standard/serialization/xml-schema-def-tool-gen

A.1.4 Structure

The main program can be separated into 3 sections.

- The first section includes “Device information”. This section provides alle needed information that are usually coming from a database of a manufacturer.
- The second section is generating the XML file. This section uses the classes generated from the XSD files and generates an XML structure. After that, the XML structure is signed using the keyfile “Signing_Private_Key.pem” that shall be located inside the output folder. The source code for signing the XML file is based on the following link: https://learn.microsoft.com/de-de/dotnet/standard/security/how-to-sign-xml-documents-with-digital-signatures
- The third section validates the signature of the file. To do this, the file as well as the out of bound provided public key is loaded. That’s why the public signing key file “Signing_Public.pem” needs to be available inside the execution folder. This part of the source code is based on the following link: https://learn.microsoft.com/en-us/dotnet/standard/security/how-to-verify-the-digital-signatures-of-xml-documents

A.2 Linux shell

A.2.1 General

Modern Server systems as well as Embedded Systems are commonly using Linux as Operating system.

Linux offers powerful tools for XML handling. Here are some examples how to handle the XML files according to this specification. The used tool is documented here:

https://manpages.debian.org/experimental/xmlsec1/xmlsec1.1

A.2.2 Signing of an XML file

For signing an existing XML file with tool xmlsec1, first there need to be the correct namespace in the Document tag of the XML file:

```
xmlns:ds="http://www.w3.org/2000/09/xmldsig#”
```

Second there needs to be a signature template integrated into the XML file (enveloped signature):

```
   <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod
        Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha256"/>
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
          <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </ds:Transforms>
        <ds:DigestMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
        <ds:DigestValue/>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue/>
    <ds:KeyInfo/>
  </ds:Signature>
```

XML files can then be signed by using the respective private key and certificate by using that command:

```
xmlsec1 --sign --privkey-pem [PRIVATE KEY,CERTIFICATE] --output
[OUTPUTFILE] [INPUTFILE]
```

A.2.3 Verification of an XML file

There are the following options for the verification of an XML file:

- Command for XML with integrated public key:

  ```
  xmlsec1 --verify [FILENAME]
  ```

- Command for XML with integrate certificate and external trusted root CA:

```
xmlsec1 --verify --trusted-pem [SIGNING ROOT CA] [FILENAME]
```

A.3 PHP

The example 1 is generated with the scripting language PHP. The applied library is "phpseclib V3.0" (https://github.com/phpseclib/phpseclib) within PHP 8.0. Source code for this implementation is not provided with this TR.

Nevertheless, here are some code snippets that might be helpful for implementers.

```
$objDSig = new XMLSecurityDSig();

// Force the reference URI being present
$objDSig->addReference($xmlDoc,XMLSecurityDSig::SHA256,
['http://www.w3.org/2000/09/xmldsig#enveloped-signature',
'http://www.w3.org/2001/10/xml-exc-c14n#'],['id_name' => 'ID',
'overwrite' => false, 'force_uri' => true]);

// Set the canonicalization method
$objDSig->setCanonicalMethod(XMLSecurityDSig::EXC_C14N);
```

As phpseclib has no direct support for setting all possible nodes of the signature one must add those nodes using native dom syntax, e.g. like this:

```
// Load the DOM node to which you want to append a new Node
$sigDoc = $objDSig->sigNode->ownerDocument;

// Add a Key Name Node
$keyNameNode=$sigDoc->createElementNS
('http://www.w3.org/2000/09/xmldsig#','ds:KeyName',‘DME_2025022
7.PEM‘);
$objDSig->appendToKeyInfo($keyNameNode);
```

A.4 Online tools

The following online tools can be used for processing different steps of this document:

- Chilkat Tools
  (https://tools.chilkat.io/xmlDsigVerify.cshtml)
  Verification of signatures of the examples

- Hex to Base64 | Base64 Encode | Base64 Converter | Base64
  (https://base64.guru/converter/encode/hex)
  Serializing the key cipher value to a base64 encoded string

- Key Wrapping
  (https://asecuritysite.com/encryption/kek)
  Wrapping of transported keys with the key encryption key

A.5 Groovy

The example 4 is generated by an application using Groovy, a scripting language with Java-like syntax. The full source code for this implementation is not provided with this TR.

Nevertheless, example scripts are provided in the Example 4 folder.

As an additional information following Java libraries were used in this implementation:

Generate XML

- org.w3c.dom
  (https://docs.oracle.com/en/java/javase/17/docs/api/java.xml/org/w3c/dom/package-summary.html)

Validate XML

- Javax.xml.validation
  (https://docs.oracle.com/en/java/javase/17/docs/api/java.xml/javax/xml/validation/package-summary.html)

Encrypt payload

- javax.crypto
  (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/javax/crypto/package-summary.html)
- Java.security
  (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/security/package-summary.html)
- Org.bouncycastle
  (https://www.bouncycastle.org/java.html)

Sign XML

- javax.xml.crypto.dsig
  (https://docs.oracle.com/en/java/javase/17/docs/api/java.xml.crypto/javax/xml/crypto/dsig/package-summary.html)

Encode

- java.util (Base64)
  (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Base64.html)

A.6 Python

The example 2 was generated using Python with the following libraries:

lxml:

- lxml — used to construct and serialise the XML document tree, building all OMSKeyExchange elements and attributes programmatically.

cryptography:

- cryptography — used for AES-256 key wrapping of the individual device keys (RFC 3394), and RSA-OAEP encryption of the Key Encryption Key (KEK) as required by the xmlenc#rsa-oaep-mgf1p algorithm identifier.

signxml:

- signxml — used to apply an enveloped XML Digital Signature (XMLDSig, RSA-SHA256) to the document, embedding the X.509 signing certificate in the signature block.

xmlschema:

- xmlschema — used to validate the generated document against the OMS XSD schema before writing the output file.

Annex B: Excerpts of OMS-S v5.1 (informative)

B.1 General

This annex provides informative tables that are not yet publicly available. They will be released and further maintained within the OMS specification version 5.1 at a later point in time. As soon as version 5.1 is released this annex becomes obsolete.

The following references are used in this technical report:

[EN 13757-7]	EN 13757-7:2025 Communication systems for meters – Part 7: Transport and security services
[OMS-S1]	Open Metering System Specification Vol. 1 – General Part, v 5.1
[OMS-S4]	Open Metering System Specification Vol. 4 – Upper Layer Protocols
[OMS-S5]	Open Metering System Specification Vol. 5 – Wireless Communication
[OMS-S6]	Open Metering System Specification Vol. 6 – OMS over 3rd party networks

B.2 Excerpt of OMS-S4

Table B.1 is going to be released within [OMS-S4] as table 5.1 and is placed here only for informative purpose.

Table B.1 – Predefined OMS-KeyIDs

Techno-logy	KeyID (Hex)	Usage	Applicable wrapper key
Symmetric	00h	Master key (MK), default communication security key	12h
	01h to 07h	Manufacturer specific communication security key [a]	
	08h	Harmonised communication security key (see Annex C, OMS-UC-07b)	12h
	09h	Service Tool communication security key	12h
	0Ah	Secondary stationary communication security key	12h
	0Bh to 0Eh	Reserved for harmonised communication security keys	12h
	0Fh	Network Operator communication security key (see [OMS-S5], 3.5)	12h
	10h	Root wrapper key; (see Figure 5.1 and Annex C, OMS-UC-08)	10h
	11h	Reserved for wrapper keys	
	12h	Communication security wrapper key (see Figure 5.1 and Annex C, OMS-UC-08)	10h, 12h
	13h	3rd party network security wrapper key (see Figure 5.1 and Annex C, OMS-UC-08)	10h, 13h

Techno-logy	KeyID (Hex)	Usage	Applicable wrapper key
	14h	Application security wrapper key; (see Figure 5.1 and Annex C, OMS-UC-08)	10h, 14h
	15h	Reserved for wrapper keys	
	16h	Reserved	
	17h	Manufacturer specific wrapper key [a]	
	18h	Authentication key for firmware image protection (See Annex C, OMS-UC-05)	14h, 18h
	19h to 1Eh	Reserved	
	1Fh	Application security key for OMS Conformance test (See Annex C, OMS-UC-00)	14h, 1Fh
	20h	Application security key for disconnection & reconnection (See Annex C, OMS-UC-03)	14h, 20h
	21h	Application security key for clock adjustment (See Annex C, OMS-UC-04b/-04c)	14h, 21h
	22h	Application security key for clock setting (See Annex C, OMS-UC-04b/-04c)	14h, 22h
	23h	Application security key for update of firmware image (See Annex C, OMS-UC-05)	14h, 23h
	24h	Application security key for OED supervision (See Annex C, OMS-UC-06)	14h, 24h
	25h	Application security key for data selection (See Annex C, OMS-UC-14)	14h, 25h
	26h	Application security key for application management (See Annex C, OMS-UC-12)	14h, 26h
	27h	Application security key for settings of an adapter (See Annex C, OMS-UC-20)	14h, 27h
	28h to 2Fh	Reserved for harmonised APL-Keys	
	30h to 4Fh	Manufacturer specific APL-Keys [a]	
	50h to 5Fh	Reserved for harmonised APL-Keys	
	60h	COSE security key for OMS over Cellular	13h
	61h to 67h	Reserved for 3rd party network keys	
	68h to 7Fh	Reserved	
Asym metric	80h to AFh	Defined in Annex E	

Techno-logy	KeyID (Hex)	Usage	Applicable wrapper key
	B0h to BFh	Reserved for harmonized Keys	
	C0h to EFh	Manufacturer specific Keys [a]	
	F0h to FFh	Reserved	
[a] These keys are under the responsibility of the manufacturer.			

B.3 Excerpt of OMS-S6

Table B.2 is going to be released within [OMS-S6] as table D.1 and is placed here only for informative purpose.

Table B.2 – Key Labels

Technology	KeyLabel	Meaning
LoRaWAN	AppKey	Root key to generate AppSKey and NwkSKey
LoRaWAN	AppSKey	Application session key
LoRaWAN	GenAppKey	Root key for LoRaWAN FUOTA protocols support
LoRaWAN	NwkSkey	Network session key
mioty	AppKey	Master key to generate AppSnKey
mioty	AppSnKey	Application session key
mioty	NwkKey	Master key to generate NwkSnKey
mioty	NwkSnKey	Network session key

NOTE: In the XML File each KeyLabel is used in a technology context. This means the identification needs only to be unique within one technology.