# ONEM2M
# TECHNICAL REPORT

| Document Number | TR-0038-V-0.5.0 |
| --- | --- |
| Document Name: | Developer guide: Implementing security example |
| Date: | 2018-03-23 |
| Abstract: | The document provides a simple use case for guiding developers to implement security when developing applications using functionalities provided by a oneM2M service platform. |

Template Version: 08 September 2015 (Dot not modify)

About oneM2M

The purpose and goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

More information about oneM2M may be found at:  http//www.oneM2M.org

# Contents

# 1 Scope

This Technical Report aims at providing guidelines to developers to implement security as specified by oneM2M TS-0003 [i.4], using a simple use case as example. It addresses the initial security provisioning for enrolment with a Service Provider, and the operational phase relying on a Security Association Establishment process to implement secure connection and access control services for basic use cases.
As example, the considered use cases are implementing a home door lock service with:-

- Authentication
- Authorisation
- Integrity
- Confidentiality

# 2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

## 2.1 Normative references

The following referenced documents are necessary for the application of the present document.

Not applicable.

## 2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] oneM2M Drafting Rules (http://www.onem2m.org/images/files/oneM2M-Drafting-Rules.pdf)

[i.2] oneM2M TS-0001: "Functional Architecture".

[i.3] oneM2M TS-0004: "Service Layer Core protocol Specification”.

[i.4] oneM2M TS-0003: "Security Solutions".

[i.5] oneM2M TS-0011: "Common Terminology".

[i.6] oneM2M TR-0025: "Application Developer Guide"

[i.7] Stefan H. Holek: "OpenSSL PKI Tutorial", Release v1.1, 13-Aug-2017

[i.8] Ivan Ristić: "OpenSSL Cookbook ", Version 1.1, Oct-2013

[i.9] OpenSSL User Manual, https://www.openssl.org/docs/manmaster/man1/ciphers.html

[i.10] oneM2M TS-0032: "MAF and MEF Interface Specification"

# 3 Definitions, symbols and abbreviations

*Delete from the above heading the word(s) which is/are not applicable.*

## 3.1 Definitions

*Clause numbering depends on applicability.*

- ***A definition shall not take the form of, or contain, a requirement.***
- ***The form of a definition shall be such that it can replace the term in context. Additional information shall be given only in the form of examples or notes (see below).***
- ***The terms and definitions shall be presented in alphabetical order.***

For the purposes of the present document, the [following] terms and definitions [given in ... and the following] apply:

### *Definition format*

**<defined term>:** <definition>

*If a definition is taken from an external source, use the format below where* [N] *identifies the external document which must be listed in Section 2 References.*

**<defined term>**[N]: <definition>

**example 1:** text used to clarify abstract rules by applying them literally

NOTE: This may contain additional information.

## 3.2 Symbols

*Clause numbering depends on applicability.*

For the purposes of the present document, the [following] symbols [given in ... and the following] apply:

### *Symbol format*

| | |
|---|---|
| <symbol> | <Explanation> |
| <2nd symbol> | <2nd Explanation> |
| <3rd symbol> | <3rd Explanation> |

## 3.3 Abbreviations

*Abbreviations should be ordered alphabetically.*

*Clause numbering depends on applicability.*

For the purposes of the present document, the [following] abbreviations [given in ... and the following] apply:

### *Abbreviation format*

| | |
|---|---|
| ARIB | Association of Radio Industries and Businesses |
| ATIS | Alliance for Telecommunications Industry Solutions |
| CCSA | China Communications Standards Association |
| ETSI | European Telecommunications Standards Institute |
| TIA | Telecommunications Industry Association, |
| TSDSI | Telecommunications Standards Development Society |
| TTA | Telecommunications Technology Association |
| TTC | Telecommunication Technology Committee |

# 4 Conventions

The key words “Shall”, ”Shall not”, “May”, ”Need not”, “Should”, ”Should not” in this document are to be interpreted as described in the oneM2M Drafting Rules [i.1]

# 5 Use case

This guide is based on a smart key use case involving front and back door locks in a home that can be remotely controlled by a user's smartphone leveraging the capabilities of oneM2M. For example, the user can remotely open the doors when friends, relatives, housekeeper or babysitter come to the user’s home. However, if a system of the use case is not secured, attackers can easily unlock the door locks by spoofing.

An overview of the use case is shown in figure 5.1-1 and the main components are introduced as follows:

- The door locks are deployed in a home and are connected to a home gateway.
- The home gateway communicates with a cloud service platform allowing the door locks to be controlled remotely by the smartphone.
- The cloud service platform supports a set of services to enable the smartphone to more easily control the door locks in the home. Some examples of services include registration, discovery, data management, group management, subscription/notification etc
- The smartphone hosts an application used to remotely control the door locks in the home and supports the following capabilities:
    - Discovery of door locks deployed in the home.
    - Sending commands to change door lock states i.e. LOCKED and UNLOCKED.
    - Retrieval of door lock states.
    - Receiving notifications when certain events occurred.
- M2M Authentication Function (MAF) is used when employing MAF-based Security Association Establishment (SAE) between field nodes and infrastructure nodes. When using Pre-Shared Key or Certificate-based SAE, the MAF is not required.

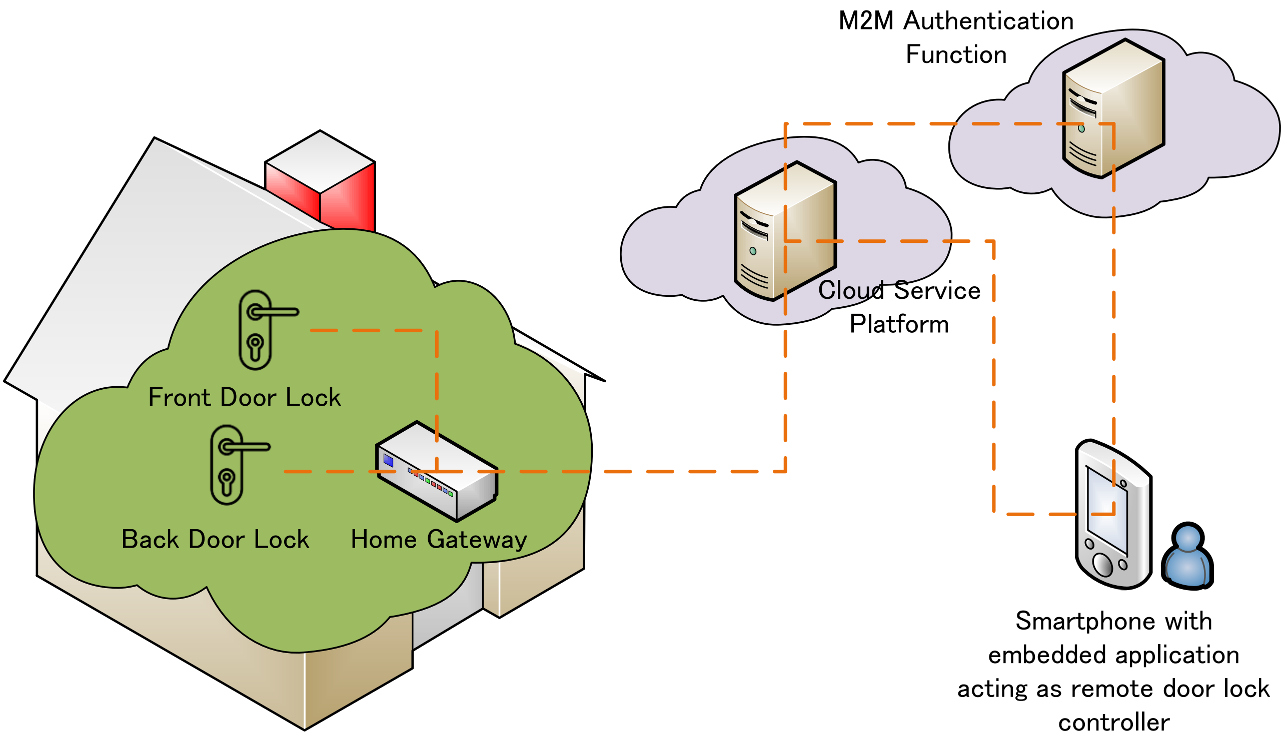


**Figure 0.1-1 Overview of remote door locks control use case**

# 6 Functional architecture

This clause describes how the different components of this use case can be represented by corresponding oneM2M architectural entities as shown in figure 6.1-1.

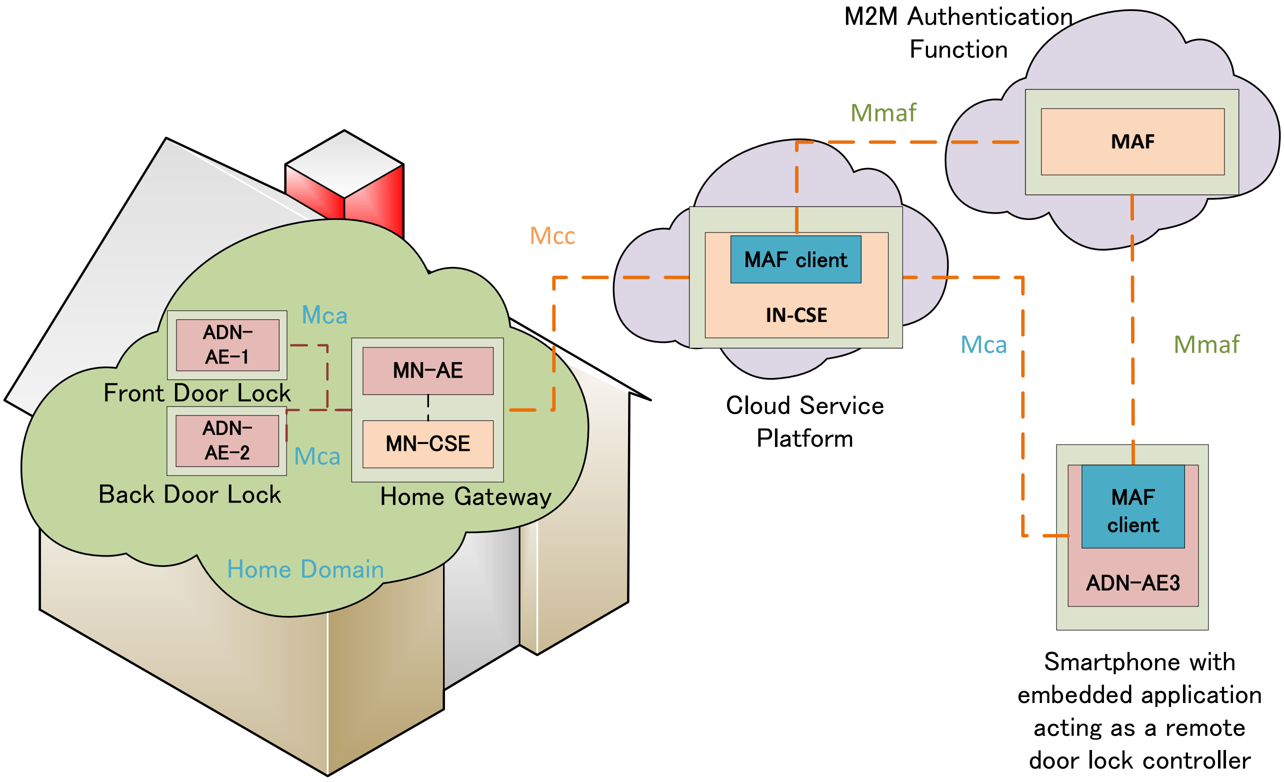


**Figure 6.1-1 oneM2M functional architecture of remote door locks control use case**

An IN-CSE is hosted in the cloud by the oneM2M Service Provider and a MN-CSE is hosted on the Home Gateway. Applications and MAF used in the current use case are classified as follows:

- ADN-AE1: an application embedded in Front Door Lock with capabilities to control Front Door Lock and interact with the home gateway MN-CSE through Mca reference point;
- ADN-AE2: an application embedded in Back Door Lock with capabilities to control Back Door Lock and interact with the home gateway MN-CSE through Mca reference point;
- ADN-AE3: an application embedded in the smartphone device with capabilities to interact directly with the oneM2M service platform IN-CSE through Mca reference point used to remotely control Front Door Lock and Back Door Lock;
- MN-AE: a gateway application embedded into the home gateway that interacts with the MN-CSE through Mca reference point.
- MAF: M2M Authentication Function assigns symmetric keys to MAF clients on the IN-CSE and the ADN-AE3 through Mmaf reference point.

# 7 Procedures and call flows

## 7.1 Security Association Establishment

### 7.1.1 Security Requirements

M2M services are offered by CSEs to AEs and/or other CSEs. To be able to use M2M services offered by one CSE, the AEs and/or CSEs need to be mutually identified and authenticated by that CSE, in order to provide protection from unauthorized access and Denial of Service attacks.

This mutual authentication enables to additionally provide encryption and integrity protection for the exchange of messages across a single Mca, Mcc or Mcc' reference point. In addition, communicating AEs that require similar protection for their own information exchanges can be provisioned to apply the same security method to their communications. This is the purpose of the Security Association Establishment (SAE) procedure.

When CoAP binding of oneM2M primitives is used, i.e. the Underlying Network communication uses UDP/IP transport, Authentication is performed by means of a DTLS Handshake.

When HTTP, MQTT or WebSocket binding of oneM2M primitives is used, i.e. the Underlying Network communication uses TCP/IP transport, Authentication is performed by means of a TLS Handshake.

For the use cases in this guideline document it is assumed that HTTP binding is employed between all applicable pairs of entities (see also TR-0025 [i.6])

In order to exemplify the use of all three Security Association Establishment Frameworks (SAEF) defined in TS-0003 [i.4] the following use cases are described:

- Provisioned Symmetric Key SAE between Door Locks and Home Gateway,
- Pre-provisioned Certificate Based SAE between Home Gateway and IN-CSE,
- MAF Based Symmetric Key SAEF between the smartphone and IN-CSE.

Communication between the MN-AE and MN-CSE internally to the Home Gateway is assumed to not require Security Association Establishment.

### 7.1.2 Provisioned Symmetric Key SAE between the Locks and the Home Gateway

In this example it is assumed that authentication between the Locks (ADN-AE1 and ADN-AE2) and the Home Gateway (MN-CSE) is performed using provisioned keys (Kpsa) and key identifiers (KpsaID).

**Configuration of ADN-AE1 and ADN-AE2:**

- The AEs are configured with the set of allowed TLS ciphersuites when using TLS-PSK as defined in clause 10.2.2 of TS-0003 [i.4]. The set of ciphersuites includes TLS_PSK_WITH_AES_128_CBC_SHA256.

- The AE is assumed to be configured with the CSE-ID of the Home Gateway which is a unique identifier within the M2M-SPs domain. The CSE-ID value is assumed as mn-cse-123456.

- The AE is assumed to be configured with a pair of credentials (psk, psk_identity) associated with the CSE-ID. An example of credential configuration is given in Table 7.1.2-1. The length of the keys Kpsa is not mandated by TS-0003 [i.4] and left to implementation. In this example the key length of 8 bytes (64 bits) is chosen. The key identifiers comply with the format specified in clause 10.5 of TS-0003 [i.4].

**Table 7.1.2-1: Example Credentials configured on ADN-AE1 and ADN-AE2**

| Entity | Kpsa (hex format) | KpsaID |
|---|---|---|
| ADN-AE1 | 1a2b3c4d5e6f7a8b | AE123456789012-Lock@in.provider.com |

| | | |
|---|---|---|
| ADN-AE2 | 12345678abcdefab | AE123456789015-Lock@in.provider.com |

**Configuration of MN-CSE (Home Gateway):**

- The MN-CSE is configured with the set of allowed TLS ciphersuites when using TLS-PSK as defined in clause 10.2.2 of TS-0003 [i.4]. The set of ciphersuites includes TLS_PSK_WITH_AES_128_CBC_SHA256.
- The MN-CSE is assumed to have a psk-lookup-table with columns for (client identity, psk, psk_identity), such that when a TLS client provides a particular psk_identity, then the MN-CSE uses the corresponding psk for establishing a TLS session, and the client identity is associated with the established TLS session. This needs to be integrated to the TLS server. Table 7.1.2-2 shows an example of credentials configured on the Home Gateway to serve ADN-AE1 and ADN-AE2, containing AE-ID, KpsaID, Kpsa. A new row would need to be added to this table for each additional AE allowed to register to the MN-CSE by using TLS_PSK.

NOTE: Some open source libraries, e.g. OpenSSL, do not provide a psk-lookup-table, but do indicate a spot in the source code where a psk-lookup could be implemented. The psk-look-up-table values could then be provided in a configuration file.

**Table 7.1.2-2: Credentials configured on MN-CSE**

| AE-ID | Kpsa (hex format) | KpsaID |
|---|---|---|
| C-lock-AE1 | 1a2b3c4d5e6f7a8b | AE123456789012-Lock@in.provider.com |
| C-lock-AE2 | 12345678abcdefab | AE123456789015-Lock@in.provider.com |

**Operation of ADN-AE1 and ADN-AE2**

When the AE is triggered to establish a TLS-PSK session with the MN-CSE using some pair (Kpsa, KpsaID), the following should occur automatically based on the AE's configuration:

- AE's TLS Client is triggered to perform a TLS-PSK handshake with the TLS values (psk, psk_identity) set to the values of (Kpsa, KpsaID), and with the configured list of TLS ciphersuites.
- On completion of the TLS handshake, the AE associates the established TLS session with the MN-CSE's CSE-ID.

**Operation of MN-CSE**

The MN-CSE' TLS Server is listening on the TLS Server port and the following should occur automatically based on the MN-CSE's configuration:

- A TLS handshake is started at the MN-CSE TLS Server on receiving a TLS handshake Client_Hello message. In the case of the AE, this includes the list of TLS-PSK ciphersuites supported by the AE for use with the MN-CSE. The MN-CSE will select a ciphersuite that is also in its configured list.
- A later TLS handshake message will include the psk_identity element set to KpsaID.
- The MN-CSE's TLS Server looks up the psk-lookup-table using KpsaID as an index, and retrieves the AE's Kpsa. If not done already, the MN-CSE queries the node's <serviceSubscribedAppRule> resource in order to check AE-ID restrictions given in the *allowedAEs* attribute. This procedure is described in clause 7.1.5.
- The MN-CSE's TLS client continues the TLS handshake with the TLS value psk set to the value of Kpsa.
- On completion of the TLS handshake, the MN-CSE associates the established TLS session with the AE's AE-ID.

Annex A provides details for implementing the TLS handshake procedure.

### 7.1.3 Certificate-based SAE between Home Gateway and IN-CSE

In this example, it is assumed that authentication between the Home Gateway (MN-CSE) and the IN-CSE is performed using CSE-ID certificates compliant with clause 10.1 of TS-0003 [i.4], which are signed by a Certification Authority (CA). The production of suitable certificates is described in Annex B.

**Configuration of MN-CSE:**

- The MN-CSE is configured with the set of allowed TLS ciphersuites when using certificates as defined in clause 10.2.3 of TS-0003 [i.4]. The set of ciphersuites includes TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256.
- The MN-CSE is assumed to be configured with a CSE-ID certificate which includes its own CSE-ID in the Subject Alternative Name (subjectAltName) field ("DNS:my.example_m2mprovider.org/mn-cse-123456"). The CSE-ID certificate is signed by a root CA certificate (in the considered example).

**Table 7.1.3-1: Example credentials configured on MN-CSE**

| Entity | Entity-ID | private key file | certificate file |
|---|---|---|---|
| MN-CSE | mn-cse-123456 | mn_cse_key.pem | 02.pem |

**Configuration of IN-CSE:**

- The IN-CSE is configured with the set of allowed TLS ciphersuites when using certificates as defined in clause 10.2.2 of TS-0003 [i.4]. The set of ciphersuites includes TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256.
- The IN-CSE is assumed to be configured with a CSE-ID certificate which includes its own CSE-ID in the Subject Alternative Name (subjectAltName) field ("DNS:my.example_m2mprovider.org/in-cse"). The CSE-ID certificate is signed by a root CA certificate. Acceptable CA certificates should be stored by the IN-CSE in a certificate store.

**Table 7.1.3-2: Example credentials configured on IN-CSE**

| Entity | Entity-ID | private key file | certificate file |
|---|---|---|---|
| IN-CSE | in-cse | in_cse_key.pem | 01.pem |

**Operation of MN-CSE**

When the MN-CSE is triggered to establish a TLS session with the IN-CSE, the following should occur automatically based on the MN-CSE's configuration:

- MN-CSE's TLS Client is triggered to perform a TLS handshake indicating its configured list of TLS ciphersuites and providing its MN-CSE certificate upon request of the TLS server to the IN-CSE.
- The MN-CSE verifies the certificate (chain) received from the IN-CSE by validating the signature(s) and by verifying that the root certificate can be trusted. Furthermore, the MN-CSE checks if the CSE-ID included in the subjectAltName field of the IN-CSEs certificate matches its configured IN-CSE ID.
- On completion of the TLS handshake, the MN-CSE associates the established TLS session with the IN-CSE's CSE-ID.

**Operation of IN-CSE**

The IN-CSE' TLS Server is listening on the TLS Server port and the following should occur automatically based on the IN-CSE's configuration:

- A TLS handshake is started at the IN-CSE TLS Server on receiving a TLS handshake Client_Hello message. In the case of the MN-CSE, this includes the list of TLS ciphersuites supported by the MN-CSE for use with the IN-CSE. The IN-CSE will select a ciphersuite that is also in its configured list.
- The IN-CSE's TLS Server is configured
    - to send its own certificate and (optional) certificate chain in a Certificate TLS handshake message
    - to request the certificate from the TLS client in a Certificate Request TLS handshake message and to validate this certificate
    - to check the CSE-ID of the MN-CSE included in the MN-CSE's certificate. If this CSE-ID is not available, then the IN-CSE obtains it from the node's <serviceSubscribedAppRule> resource.
- On completion of the TLS handshake, the IN-CSE associates the established TLS session with the MN-CSE's CSE-ID.

### 7.1.4 MAF-based SAE between Smartphone and IN-CSE

In this example, it is considered the case where the AE implemented on a smartphone registers to the IN-CSE using MAF-based SAE.

It is assumed that the MAF client, associated with ADN-AE3 and implemented on the smartphone, is configured to use certificate-based SAE when communicating with the MAF. The MAF Client of the IN-CSE is assumed to be already registered with the MAF. The security association between AE1 and the IN-CSE is then established as illustrated in figure 7.1.4-1 with the steps described below. The communication between MAF clients and the MAF is assumed to comply with the MAF interface specification TS-0032 [i.10], where HTTP is used as binding protocol. JSON serialization of primitives is employed.

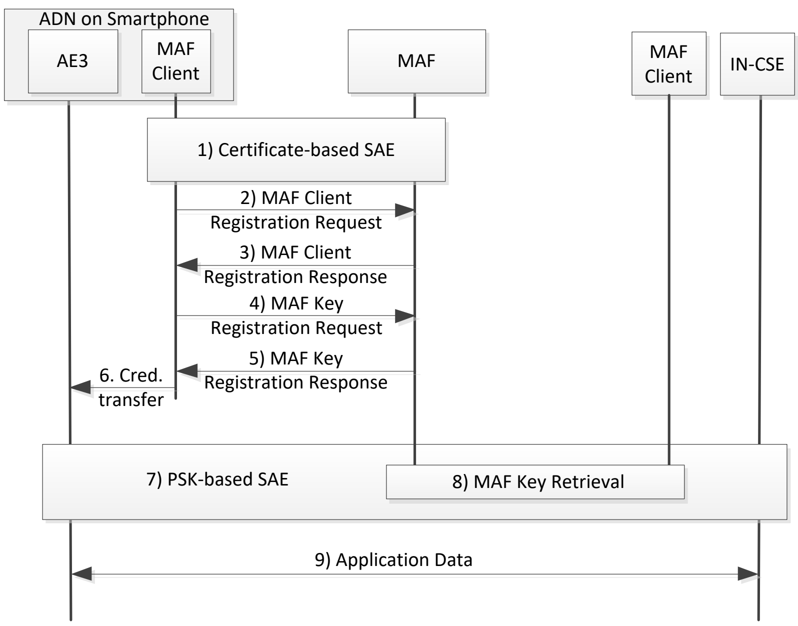


**Figure 7.1.4-1: MAF-Based Security Association Establishment**

1. A security association between the MAF client and the MAF is established. This procedure is the same as described in clause 7.1.4 and Annex A.3. In this example it is assumed that keying material to be used later on in the security association between ADN-AE3 and IN-CSE is derived at both ends using the TLS key exporter function (see clauses 8.2.2.3 and 8.3.5.3.7 of TS-0003 [i.4]). Further details of this procedure are described in Annex A.4.

    *Editor's note: When a MAF client is associated with a single AE or CSE, an already existing AE-ID or CSE-ID certificate may be used in the TLS handshake. This would require some clarifications in TS-0003. TS-0003 currently mandates the use of a device certificate, which requires a device ID in subjectAltName.*

2. The MAF client registers to the MAF by sending a MAF client registration request as specified in clause 8.8.2.3 of TS-0003 [i.4]:

| JSON serialized primitive | Comments |
|---|---|
| {<br>"op": 1,<br>"to": "//myMAF.provider.org/-/",<br>"fr": "0 2 481 1 100 3030 10011",<br>"rqi": "0001",<br>"ty": 3,<br>"pc": {"sec:macr": {<br>"et": "20181113T110000",<br>"adfq": "mytrustenabler.org"<br>}},<br>"rcn": 7<br>} | (request primitive)<br>operation = CREATE<br>to = default MAFBase address<br>from = device id of device where MAF client is installed<br>request identifier, assigned by originator<br>resource type = <mafClientReg> to be created<br>content = global element name of <mafClientReg><br>expirationTime = 2018-11-13 11:00:00 UTC<br>adminFQDN<br><br>result content = Original Resource |

3. The MAF sends the response to the MAF client:

| JSON serialized primitive | Comments |
|---|---|
| {<br>"rsc": 2001,<br>"rqi": "0001",<br>"pc": {"sec:macr": {<br>"rn": "MACR000001",<br>"ty": 3,<br>"ri": "macr000001",<br>"pi": "mb01",<br>"ct": "20171113T110000",<br>"lt": "20171113T110000",<br>"et": "20181113T110000",<br>"cr": "0 2 481 1 100 3030 10011",<br>"adfq": "mytrustenabler.org",<br>"aski":<br>"FF15D84E3E38D6974B0EB3E5606C85FE@myMAF.provider.org"<br>}}<br>} | (response primitive)<br>response status code, CREATED<br>request identifier<br>content=global element name <mafClientReg><br>resource name, assigned by MAF<br>resource type = <mafClientReg><br>resource identifier, assigned by MAF<br>parent identifier, resource id of MAFBase<br>creation time<br>last modified time<br>expiration time, 1 year after creation<br>creator, MAF client id<br>adminFQDN, fqdn of trust enabler<br>key identifier |

4. MAF key registration request as described in clause 8.8.2.7 of TS-0003 [i.4].

| JSON serialized primitive | Comments |
|---|---|
| {<br>"op": 1,<br>"to": "//myMAF.provider.org/-/macr000001",<br>"fr": "0 2 481 1 100 3030 10011",<br>"rqi": "0002",<br>"ty": 5,<br>"pc": {"sec:mkr": {<br>"et": "20171120T110000",<br>"adfq": "mytrustenabler.org",<br><br>"suid": 11<br>}},<br>"rcn": 7<br>} | (request primitive)<br>operation = CREATE<br>to = address of <mafClientReg> parent resource<br>from = device id of MAF client (= MAF client ID)<br>request identifier, assigned by originator<br>resource type = <symmKeyReg> to be created<br>content = global element name of <symmKeyReg><br>expiration time, 1 week after creation<br>adminFQDN, fqdn ofd trust enabler<br><br><br>security usage id = MAF-based SAEF<br><br>result content = Original Resource |

5. MAF key registration response. Note that the *keyValue* attribute is not returned to the MAF client as this key is derived from the TLS key exporter function.

| JSON serialized primitive | Comments |
|---|---|

| | |
|---|---|
| {<br>"rsc": 2001,<br>"rqi": "0002",<br>"pc": {"sec:mkr": {<br>"rn": "SK00001",<br>"ty": 5,<br>"ri":<br>"FF15D84E3E38D6974B0EB3E5606C85FE",<br>"pi": "macr000001",<br>"ct": "20171113T110001",<br>"lt": "20171113T110001",<br>"et": "20171120T110001",<br>"cr": "0 2 481 1 100 3030 10011",<br>"adfq": "mytrustenabler.org",<br>"suid": 11,<br>"tgis": "//my.m2mprovider.org/in-cse"<br>}}<br>} | (response primitive)<br>response status code, CREATED<br>request identifier<br>content=global element name <symmKeyReg><br>resource name, assigned by MAF<br>resource type = <symmKeyReg><br>resource identifier, assigned by MAF equal to relativeKeyID, see Annex A.4<br>parent identifier, resource id of <mafClientReg><br>creation time<br>last modified time<br>expiration time, 1 week after creation<br>creator, MAF client id<br>adminFQDN, fqdn of trust enabler<br>security usage id = MAF-based SAEF<br>list of target identifiers, registrar CSE id<br>Note: key value is not returned to MAF client in this procedure |

6. Using the keying material established in step 1 the security credentials psk and psk_identity are transferred from the MAF client to the AE (see Annex A.4 for more details).
7. PSK-based security association is established between AE3 and the IN-CSE, as described in clause 7.1.3 and Annex A.2 using psk and psk_identity from step 6.
8. As part of step 7), the MAF client associated with the IN-CSE retrieves the PSK credential from the MAF which is identified from the fqdn-part of the psk_identity value by means of triggering a MAF Key Retrieval procedure as specified in clause 8.8.2.8 of TS-0003 [i.4]. It is assumed that a security association between IN-CSE's MAF client and the MAF already exists prior to execution of the MAF Key Retrieval procedure. The Key Retrieval request and response primitives are shown in the Table below:

| JSON serialized primitive | Comments |
|---|---|
| {<br>"op": 2,<br>"to": "//myMAF.provider.org/-/FF15D84E3E38D6974B0EB3E5606C85FE",<br>"fr": "//my.m2mprovider.org/in-cse",<br>"rqi": "ABC28F",<br>"rcn": 7<br>} | (request primitive)<br>operation = RETRIEVE<br>to = address of <symmKeyReg> parent resource = KcID<br><br>from = IN-CSE identifier<br>request identifier, assigned by originator<br>result content = Original Resource |
| {<br>"rsc": 2000,<br>"rqi": "ABC28F",<br>"pc": {"sec:mkr": {<br>"rn": "SK00001",<br>"ty": 5,<br>"ri":<br>"FF15D84E3E38D6974B0EB3E5606C85FE",<br>"pi": "macr000001",<br>"ct": "20171113T110001",<br>"lt": "20171113T110001",<br>"et": "20171120T110001",<br>"cr": "0 2 481 1 100 3030 10011",<br>"adfq": "mytrustenabler.org",<br>"suid": 11,<br>"tgis": "//my.m2mprovider.org/in-cse",<br>"kv":<br>"37F61D5A7FEA1E9CFD8DB76D2F8B6230130EF8A84F9F9F967DA385867984EED0"<br>}}<br>} | (response primitive)<br>response status code, OK<br>request identifier<br>content=global element name <symmKeyReg><br>resource name, assigned by MAF<br>resource type = <symmKeyReg><br>resource identifier = relative Key id<br><br>parent identifier, resource id of <mafClientReg><br>creation time<br>last modified time<br>expiration time, 1 week after creation<br>creator, MAF client id<br>adminFQDN, fqdn of trust enabler<br>security usage id = MAF-based SAEF<br>list of target identifiers, registrar CSE id<br>key value, as derived with TLS key material exporter function |

9. Encrypted messages can be exchanged between AE3 and the IN-CSE.

## 7.1.5 Registration upon successful SAE

An AE or CSE which has not registered to its registrar CSE yet, is assumed to be pre-configured such that it attempts to perform a registration procedure right after the device is powered on and after it has established network connectivity. In this case the first request primitive sent by an AE or CSE entity via Mca or Mcc interfaces is either a “Create <*AE*>” or “Create <*remoteCSE*>” request primitive, respectively.

When an AE registers, the registrar CSE needs to retrieve and check the service subscription information which is defined in a <*m2mServiceSubscriptionProfile*> instance on the IN-CSE (see clause 10.2.2.2 of TS-0001 [i.2]).

NOTE: In the present Release, <serviceSubscribedAppRule> does not allow to validate the association between CSE registrees and their applicable credential identifiers when registering to their registrar CSE.

For the use case example illustrated in figure 6.1-1, the overall structure of service subscription information can look as shown in figure 7.1.5-1. It is assumed that these resources have been configured on the IN-CSE prior to the registration procedure. Their creation is out of scope of the present document.

The instance of a <*m2mServiceSubscriptionProfile*> with its children and linked resources as shown in figure 7.1.5-1 includes all information exposed on the Mcc interface related to a service subscription of the subscriber who owns and operates the considered example home network in figure 6.1-1. An <*m2mServiceSubscriptionProfile*> resource does not include any resource-specific attributes itself. It acts as parent of all <*serviceSubscribedNode*> resources related to a specific subscriber. Every node shown in figure 6.1-1, i.e. Front Door Lock, Back Door Lock, Smartphone, Home Gateway and Cloud Infrastructure, can have an associated instance of a <*serviceSubscribedNode*> child resource configured. However, the <*serviceSubscribedNode*> resource of an ADN only includes the *nodeID* attribute, which is relevant for Device Management procedures but irrelevant in the context of the registration procedure. Therefore figure 7.1.5-1 shows <*serviceSubscribedNode*> resources related to the MN and IN only. These include in addition to the *nodeID* attribute a *CSE-ID* and a *ruleLinks* attribute. The *CSE-ID* relates to the CSE of the node identified by the *nodeID* attribute.

The *ruleLinks* attribute assign <*serviceSubscribedAppRule*> resources to a <*serviceSubscribedNode*> resourcs. (in terms of a list of their *resourceID* values). In the specific example considered here, it is assumed that there is one <*serviceSubscribedAppRule*> resource instance configures for each AE which is allowed to register to a given CSE.

In the example considered in figure 6.1-1, the Home Gateway (MN) hosts three registree AEs: ADN-AE1, ADN-AE2 and MN-AE. Therefore, the <*serviceSubscribedNode*> resource associated with the Home Gateway could have 3 different <*serviceSubscribedAppRule*> resources assigned, one for each AE shown in figure 6.1-1. The service subscriber employs ADN-AE3 as door lock controller which registers to the IN-CSE directly. The resource tree in figure 7.1.5-1 therefore also includes a <*serviceSubscribedNode*> resource associated with the IN. This <*serviceSubscribedNode*> reveals *nodeID* and *CSE-ID* of the IN-CSE and it is assumed to have a *ruleLink* attribute which includes the resource identifier of a <*serviceSubscribedAppRule*> resource which includes information related to ADN-AE3.

The <*serviceSubscribedAppRule*> resource can have 3 specific attributes: *allowedCredIDs*, *allowedAppIDs* and *allowedAEs*. Each of these attributes generally can include a list of elements. If a <*serviceSubscribedAppRule*> relates to a single AE only, the *allowedAppIDs* and *allowedAEs* attributes contain a single element only.

Table 7.1.5-1 shows a suitable setting of these attributes for each of the three <*serviceSubscribedAppRule*> resources.

For instance, the column with heading ADN-AE1, shows the attributes of the <*serviceSubscribedAppRule*> resource which relates to ADN-AE1. In this case, the *allowedAEs* attribute includes the AE-ID stem to be assigned to ADN-AE1 by the registrar MN-CSE, which is of the form as used in the example in clause 7.1.2. The wildcard part is substituted by the MN-CSE. The *allowedAppIDs* attribute includes the App-ID and the *allowedCredIDs* attribute includes security credential identifiers applicable for ADN-AE1. The columns with headings ADN-AE2 and ADN-AE3 of Table 7.1.5-1 shows the set of applicable parameters for those respective AEs.

At <AE> registration, information included in applicable <*serviceSubscribedAppRule*> resources is examined by the registrar CSE and compared if it matches with security credentials employed for Security Association Establishment (SAE), and App-ID and AE-ID indicated in the registration request message.

In case the information used by the registree does not match with the information given in applicable <*serviceSubscribedAppRule*> resources, the registration request needs to be rejected by the registrar CSE.

Note that if no applicable <*serviceSubscribedAppRule*> resources are configured, all registration requests passing Security Association Establishment successfully can be granted by the registrar.

Figure 7.1.5-2 outlines the message and processing flow related to Security Association Establishment as described in clauses 7.1.2, 7.1.3 and 7.1.4 for ADN-AE1 and ADN-AE2, MN-CSE and ADN-AE3, respectively, and the subsequent registration procedures, where service subscription information is evaluated. The description under the figure describes each step of the message and processing sequence.

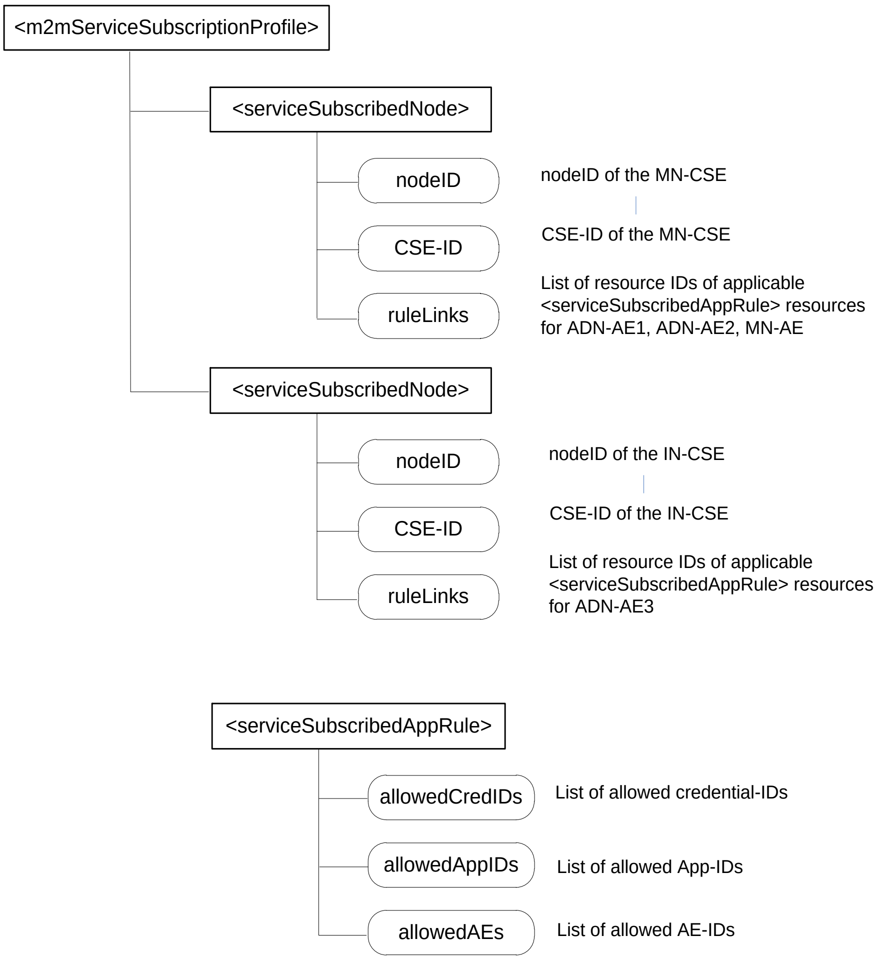


**Figure 7.1.5-1: Service subscription information stored on the IN-CSE for the use case in Fig. 6.1-1**

**Table 7.1.5-1: Value setting of <serviceSubscribedAppRule> attributes**

| Attribute | ADN-AE1 | ADN-AE2 | ADN-AE3 |
|---|---|---|---|
| *allowedCredIDs* | 12-AE123456789012-Lock@in.provider.com | 12-AE123456789015-Lock@in.provider.com | 32-*@myMAF.provider.org |
| *allowedAppIDs* | doorlock-123 | doorlock-123 | lockControl-ABC12 |
| *allowedAEs* | C-lock-AE* | C-lock-AE* | C-lockControl-AE* |

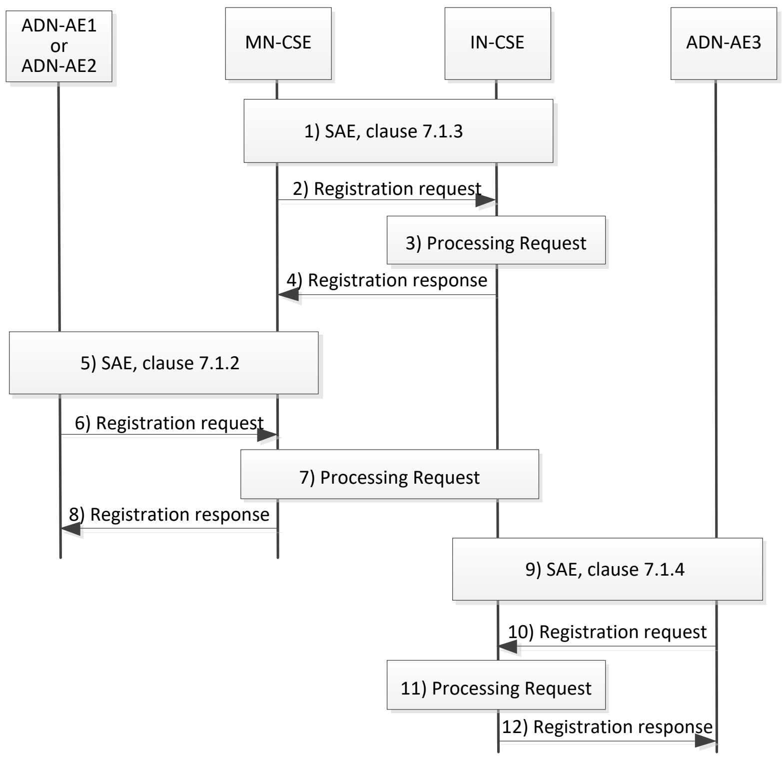


**Figure 7.1.5-2: Message sequence of Security Association Establishment and registration procedures**

1) A Security Association between MN-CSE and IN-CSE is established as described in clause 7.1.3 using public key certificates.
2) The MN-CSE sends a registration request message to the IN-CSE. Note that the MN-CSE registers to the IN-CSE before any AEs can register to the MN-CSE.
3) The IN-CSE checks the content of the registration request message (i.e. create <remoteCSE> request) as specified in clause 10.2.2.6 of TS-0001 (Rel-3) and clause 7.4.4 of TS-0004 (Rel-3).
4) The registrar IN-CSE replies with a Registration response message. For the following steps it assumed that the MN-CSE registration was successful (Response Status Code 2001 “CREATED”).
5) ADN-AE1 (and ADN-AE2) establish a security association with the MN-CSE using the procedure described in clause 7.1.2, using symmetric key credentials.
6) The AE sends a registration request message to the MN-CSE. The registration request may or may not include an AE-ID in the ***From*** parameter and in the <AE> resource representation included in the ***Content***.

7) The AE registration procedure in clause 10.2.2.2 of TS-0001 (Rel-3) defines different processing cases depending in what information about AE-ID is provided with the request message. Here, it is assumed that the registering node already has an AE-ID preconfigured. Also the App-ID of the <AE> resource is indicated in the ***Content*** of the request message. In this step, the MN-CSE needs to check if there is a <*serviceSubscribedNode*> resource configured on the IN-CSE applicable to the MN-CSE, i.e. a resource instance which includes the CSE-ID assigned to the MN-CSE. This information can be obtained with a filtered retrieve request sent by the MN-CSE to its registrar IN-CSE, as described in clause 10.2.2.2 of TS-0001 (Rel-3). Once retrieved, the MN-CSE needs to retrieve any applicable <*serviceSubscribedAppRule*> resources as indicated in the *ruleLinks* attribute of the <*serviceSubscribedNode*> resource. In the example considered here, the MN-AE retrieves the <*serviceSubscribedAppRule*> with the setting for ADN-AE1 (or ADN-AE2) as shown in table 7.1.5-1. It then compares whether or not:

    (i) the AE-ID given in the *allowedAEs* attribute matches the AE-ID given in the registration request,

    (ii) the App-ID given in the *allowedAppIDs* attribute matches the App-ID given in the ***Content*** of the request,

    (iii) the credential-ID given in the *allowedCredIDs* attribute matches the security credential which has been used in the SAE procedure (in this example the symmetric key credential derived from KpsaID as shown in table 7.1.2-1). A credential-ID included in the *allowedCredIDs* attribute is comprised of two parts:

    - a credential-ID type identifier (CredIDTypeID, defined in Table 12.3.2.1-1 of TS-0003 [i.4]. In this example CredIDTypeID = 12 indicates that PSK-based SAE is used.
    - a specific identifier of the allowed security credential, which is KpsaID for the given CredIDTypeID. The format of KpsaID is defined in clause 10.5 of TS-0003 [i.4].

8) If any of the above checks fails, the registration request is rejected with a respective error response. If the AE indicates the AE-ID and App-ID as given in the applicable <*serviceSubscribedAppRule*> the registration request can be granted with a successful response (Response Status Code 2001 "CREATED").

9) ADN-AE3 establishes a security association with the MN-CSE using the procedure described in clause 7.1.4, using MAF-assigned symmetric key credentials. Note that this step and the subsequent registration procedure is independent of the previous steps and can occur at any time within the message sequence.

10) The ADN-AE3 sends a registration request message to the IN-CSE which is assumed to include preassigned AE-ID and App-ID.

11) Similarly, as in step 7), the IN-CSE evaluates the registration requests. Since the IN-CSE is the host of any service subscription related resource, if configured, it is locally available and does not need to be retrieved via the Mcc interface. The IN-CSE performs the same checks between AE-ID and *allowedAEs*, AppID and *allowedAppIDs* and credential-IDs used in the SAE procedure and *allowedCredIDs* and as described in step 7)

12) If any of the above checks fails, the registration request is rejected with a respective error response. If the AE indicates AE-ID and App-ID as given in the applicable <*serviceSubscribedAppRule*> the registration request can be granted with a successful response (Response Status Code 2001 "CREATED").

Subsequent to successful registration, an AE can send any other request primitives. In such transactions, the receiver of any request message can perform a procedure denoted *AE impersonation prevention* (see clause 7.2 of TS-0003 [i.4]). For each received request message, the receiver checks if the AE-ID in the ***From*** parameter is associated with the credentials used for security association establishment.

## 7.2 Authorisation

### 7.2.1 Introduction

The Authorization function is responsible for controlling access to resources and services hosted by CSEs and AEs.

The authorization procedure requires that the originator of the resource access request message has been identified to the Authentication function, and originator and receiver are mutually authenticated with each other. Mutual authentication between adjacent entities, i.e. between registree and registrar, can be ensured by the Security Association Establishment procedures as described in clause 7.1.

In the oneM2M system, access to resources can be controlled by assignment of access control policies to the resources. Access control policies govern *who* (originators) can do *what* (operations) under *which* circumstances (context information associated with a request).

Access control policies can be configured in the form of <*accessControlPolicy*> resources (ACP) which are statically assigned to other resources by means of an *accessControlPolicyID* attribute. The *accessControlPolicyID* attribute can include a list of resource identifiers of <*accessControlPolicy*> resources which include the access control rules applicable to that resource. This is illustrated in Figure 7.2.1-1. The links refer to the elements included in the *accessControlPolicyID* attribute. Each configured <*accessControlPolicy*> resource ACP1…3 includes one or more ACP rule(s). Each such ACP rule *who* can do *what* under *which* circumstances.

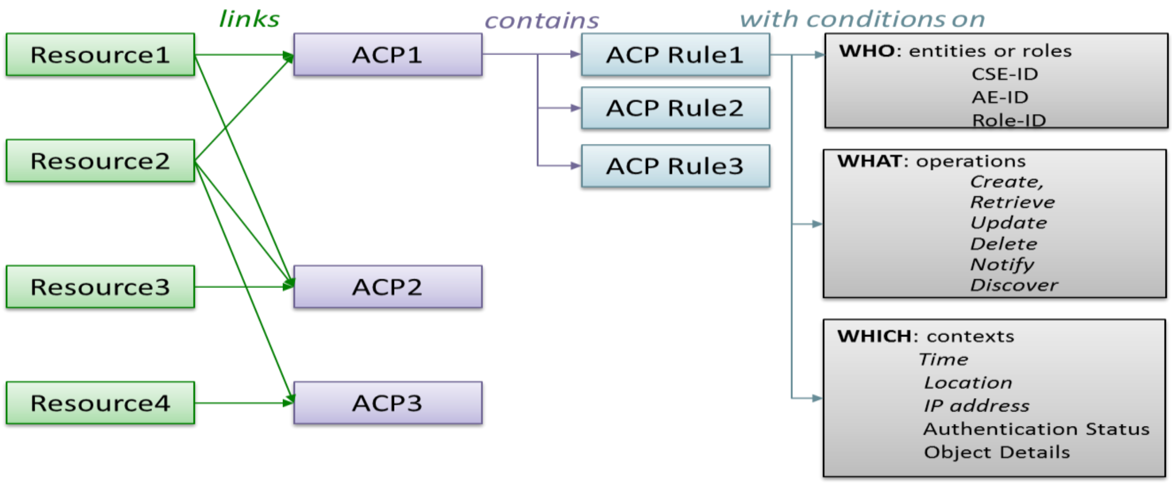


**Figure 7.2.1-1: Assignment of Access Control Policies (ACP) to resources**

The details of access control policy information and the access control mechanism are specified clause 7.1. of TS-0003 [i.4].

This clause focuses on a simple example of configuring access control policy information adequate for the considered use case.

More advanced access control mechanisms, which employ dynamic access control, role-based access control and distributed access control are not addressed in the present version of this document.

### 7.2.2 Resource structure of the example use case

Figure 7.2.2-1 shows an example resource tree hosted by the MN-CSE which is suitable for the door lock use case as described in clauses 5 and 6.

The <*AE*> resources representing the two door locks are created at registration. The resource tree under each of these <*AE*> resources looks the same. Therefore, the figure exemplifies only the resource structure under ADN-AE1. After completion of the <AE> registration procedure it is assumed that following procedures are executed by each door lock:

1. Creation of a <*container*> resource representing the state information of the respective door lock;
2. Creation of a first <*contentInstance*> resource, which includes the actual door lock state (i.e. “locked” or “unlocked”) in the *content* attribute, e.g. in the form of a binary representation;
3. Creation of a <*pollingChannel*> resource to be employed by the door lock AE;
4. Creation of a <*subscription*> resource under the <*container*> resource, which defines conditions for which a notification is sent to the respective door lock application;
5. Creation of another <*subscription*> resource which defines conditions for which a notification is sent to the door lock controller application. This resource is created by the door lock controller (see below).

Note that the detailed procedures to create the above resources are not in the scope of the present document. These procedures are described in the Applications Developer Guide TR-0025.

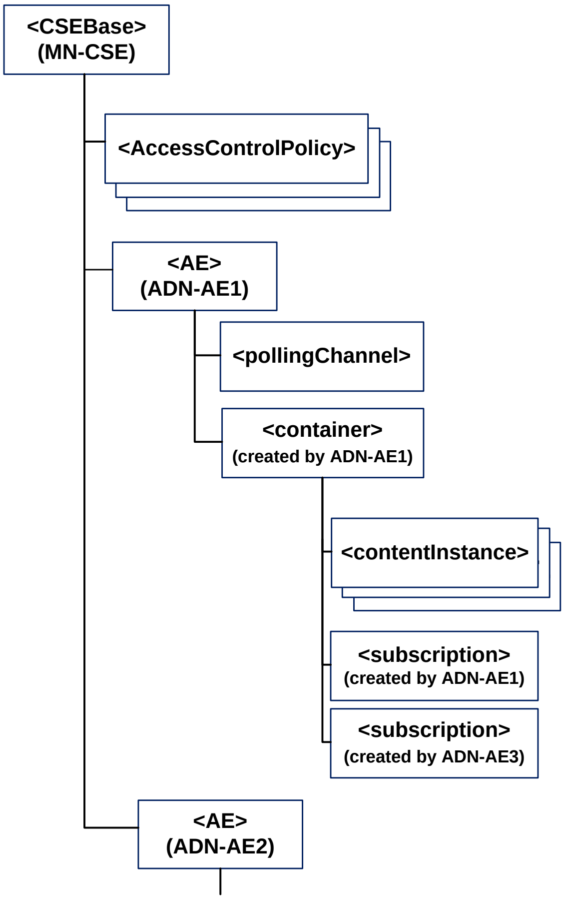


**Figure 7.2.2-1: Resource tree hosted by the MN-CSE**

The door lock controller ADN-AE3 implemented on the smartphone registers to the IN-CSE. The created <*AE*> resource does not require <*container*> child resources for its function. It is assumed in this example that ADN-AE3 is not request reachable and therefore requires also a <*pollingChannel*> child resource. In this case, after completion of the <*AE*> registration procedure, ADN-AE3 is assumed to execute following procedures:

1. Creation of a <*pollingChannel*> resource to be employed to retrieve the <*pollingChannelURI*> virtual child resource;
2. Creation of the <*subscription*> resource under each of the <*container*> resources of ADN-AE1 and ADN-AE2. This <*subscription*> resource defines conditions for which a notification is sent to the door lock controller ADN-AE3.

### 7.2.3 Configuration of <*accessControlPolicy*> resources

The resource types defined by the oneM2M specifications can be broadly categorized into two classes:

a) Resource types which have an optional *accessControlPolicyID* attribute. These are denoted as "regular resource types" in the following (cf. clause 6.5 of TS-0004).
b) Resource types which do not have an optional *accessControlPolicyID* attribute. These are denoted as "subordinate resource types" in the following (cf. clause 6.5 of TS-0004).

Access control to subordinate resource types is specified on a case-by-case basis for each individual resource type in TS-0001. The <*accessControlPolicy*> and <*pollingChannel*> belong into this category.

Resources of type <*accessControlPolicy*> include a *selfPrivileges* attribute which defines access privileges to change an <*accessControlPolicy*> itself.

Resources of type <*pollingChannel*> are accessible by the creator of each resource instance only.

For "regular resource types" which do not have any *accessControlPolicyID* attribute assigned yet, default access privileges apply.  The default access privilege gives the creator unrestricted access to the resource, i.e. it allows the creator of the resource to execute all possible operations defined for that resource type.

Access control management of "regular resource types" generally consists of two steps:

1. Creation of suitable <*accessControlPolicy*> resources
2. Setting of the *accessControlPolicyID* attribute in applicable resources

When an <*AE*> resources is created at AE registration, access control policies do not apply. Authorization is done solely based on M2M service subscription information, as outlined in clause 7.1.5.

Thanks to the default access privilege, the originator/creator of the <*AE*> resource is allowed to create child resources as well as children of children. This means, the resource tree shown in Figure 7.2.2-1 under the <*AE*> resource of ADN-AE1 or ADN-AE2 can be created without any <*accessControlPolicy*> resources assigned in the *accessControlPolicyID* attribute.

However, when originators other than the creator of the <*AE*> resource need to be given access, then access control policies are assigned.  For the use case example considered here, at least access control policies areconfigured which allow the door lock controller ADN-AE3 to update and retrieve the <*container*> resources created by the door lock applications ADN-AE1 and ADN-AE2 and to create a <*subscription*> to these containers.

An <*accessControlPolicy*> resource contains two mandatory resource-specific attributes, denoted *privileges* and *selfPrivileges*.  Each of these attributes includes one or more *access control rule(s)*. An access control rule has two mandatory elements, namely *accessControlOriginators* and *accessControlOperations*. In addition, there can be up to three optional elements, denoted *accessControlContexts*, *accessControlAuthenticationFlags*, and *accessControlObjectDetails*.

It is focused on the mandatory elements of an access control rule first. The *accessControlOriginators* element of an access control rule represents a list of originators (i.e. AE-IDs or CSE-IDs) which are allowed to perform operations defined in the *accessControlOperations* element. See clause 7.1.3 and Table 7.1.3-1 in TS-0003 [i.4] for a detailed description of the elements of access control rules. TS-0004 defines how the values of elements and sub-elements are represented in terms of XML schema datatypes.

An example representation of the *privileges* and *selfPrivileges* attributes equivalent with what is denoted as "default access privilege" to resources created by C-lock-AE1 looks as follows (in XML format with long names for better readability):

```
<privileges>
     <accessControlRule>
          <accessControlOriginators>C-lock-AE1</accessControlOriginators>
          <accessControlOperations>63</accessControlOperations>
     </accessControlRule>
</privileges>
<selfPrivileges>
     <accessControlRule>
          <accessControlOriginators>C-lock-AE1</accessControlOriginators>
          <accessControlOperations>63</accessControlOperations>
</selfPrivileges>
```

The term default access privilege is defined in clause 9.6.1.3.2 of TS-0001 [i.2]. It enables the creator of a resource to apply all applicable of operations on it. Note that once explicit access privileges are assigned to a resource in the *accessControlPolicyID* attribute, the "default access privilege" does not apply anymore. If the default access privilege should remain in place, it needs to be defined explicitly and made part of the applicable set of access control rule (either as a separate <*accessControlPolicy*> resource, or as a specific access control rule which is included with other rules into an <*accessControlPolicy*> resource).

The *accessControlOriginators* element of an access control rule is represented as a list of members which can a type as given in table 7.2.3-1.

**Table 7.2.3-1: Types of *accessControlOriginators* element**

| Member Type | Criterion to pass this constraint |
|---|---|
| *SP Domain name* | FQDN of a service provider's domain, e.g. area10023.myprovider.org. All AEs and CSEs in this domain are granted access within the *accessControlOriginators* constraints |
| *originatorID* | a) CSE-ID, AE-ID, wildcard character '*' allowed.<br>b) resource-ID of a <*group*> resource that contains the AE or CSE representing the originator, no wildcard allowed.<br>Originator of the request which matches the given CSE-ID or AE-ID is granted access within the *accessControlOriginators* constraints |
| *Key word "all"* | Any Originators are allowed to access the resource within the *accessControlOriginators* constraints |
| *Role-ID* | a) Role Identifier associated with an AE /AE-ID as defined in *allowedRole-ID* attribute of <*serviceSubscribedAppRule*><br>b) Role identifier associated with an AE /AE-ID as defined in a <*role*> resource<br>Example Role-ID: 1234abcd@role-issuer.com |

The *accessControlOperations* element of an access control rule is represented as decimal number in the range of 1 … 63 which represents an encoded combination of permitted operations on the resource. The encoding is defined in table 7.2.3-1.

When converting the decimal number into a 6-bit binary representation, each binary digit corresponds to one specific operation as illustrated in Table 7.2.3-2. A digit with value 1 or 0 means that the respective operation is allowed or disallowed, respectively. In other words, the digit "1" represents a flag that the corresponding operation is permitted.

**Table 7.2.3-2: Representation of *accessControlOperations* parameter**

| Enumeration | Discov. | Notify | Delete | Update | Retrieve | Create |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 |
| … | … | .. | … | … | … | … |
| 63 | 1 | 1 | 1 | 1 | 1 | 1 |

For example, if CRUD operations are allowed and Notify and Discovery disallowed, the value of *accessControlOperations* parameter needs to be set to 15 (binary: 001111).

CRUD and Discovery represent operations which are executed on the resource addressed in the ***To*** parameter of a request primitive. A Notify request message, however, does not represent an operation on a resource.

A Notify request message (aka. Notification) is typically sent to an entity (AE or CSE) to inform it, that a special event has occurred which the receiver of the Notification has subscribed to by means of a <*subscription*> resource.

Other use cases for Notifications include the transfer of the response primitive in reply to a request which is sent in non-blocking asynchronous transmission mode and the response to long polling (i.e. Retrieve request targeting at a <*pollingChannelURI*> virtual resource).

See clause 7.5.1.2 of TS-0004 [1.3] for a comprehensive description of Notification use cases.

Notify request primitives are sent to the entity which is identified by the ***To*** parameter and denoted as notification target.  Notifications which are triggered by conditions defined in a <*subscription*> resource are sent to the notification target(s) given in the *notificationURI* attribute of the <*subscription*> resource.   *notificationURI* attribute is represented as a list which can include one or more members. The applicable formats of each member of this attribute are specified in clause 9.6.8 of TS-0001 [i.2].

Notification targets are represented as a oneM2M resource-ID which can be represented in various formats as defined in clause 7.2 of TS-0001 [i.2].

The Notify “flag” in the *accessControlOriginators* element is validated for every Notify request message sent to either an AE or CSE. The notification target, i.e. the ***To*** parameter of a Notify request primitive is represented in the form of a resource-ID of an <*AE*> or <*remoteCSE*> resource. The Notify “flag” in the *accessControlOriginators* element of the <*AE*> or <*remoteCSE*> associated with the notification target is set to pass this access control condition. The Notify “flag” indicates that the respective entity is allowed to receive Notify request messages.

There are several implementation options how to setup access control in a oneM2M system. If these resources are to be created and managed in a standard compliant way, the natural approach is to employ an AE for this purpose. This could be a special AE just serving the purpose of managing access control, or it could be implemented as an additional function of an AE which also serves other purposes.

The oneM2M standard is not mandating a specific mechanism how to configure access control policies. The Privacy Policy Manager (PPM) concept described in clause 11 of TS-0001 [i.4] represents one approach which employs an IN-AE service provided by an application services provider.

The following design options can be considered in the context of the door lock use case:

1) Develop a separate AE which registers to the MN-CSE directly. This could be either a separate ADN-AE or an MN-AE, i.e. an AE residing on the same device as the MN-CSE.
2) Develop a separate AE which registers to the IN-CSE and which can access the MN-CSE. In this case it could be implemented either as integral part of the door lock controller ADN-AE3 or it could be implemented as a separate additional application which runs on the same ADN (smartphone) as ADN-AE3.
3) The AE employed for setting of access control policies is an IN-AE managed by an M2M service provider. In this case management of access control policies is executed under responsibility of the M2M service provider based on some agreement between the end user and service provider.
4) The AE may function in a fully automated manner or in a semi-automated manner requiring manual interaction by a human user. If the latter case is desired, it would be useful if the device hosting the AE has capability to provide a rich graphical user interface (e.g. such as a personal computer or a smart phone).

In the following it is considered the implementation of an AE which exclusively serves configuration of access control policies. Such AE could be deployed flexibly on different M2M devices in accordance with a user’s preference. For the use case considered in the present document, it is assumed that this AE is collocated with the MN-CSE on the Home Gateway (MN). In the following this AE is denoted MN-AE and it is assigned the AE-ID “C-ACP-mgr”.

For the considered door lock use case, MN-AE provides the following basic functionality:

- Discovery of any AEs associated with the given example door lock service
- Interpretation of the function of each discovered AE (e.g. from App-ID)
- Creation of <*accessControlPolicy*> resources on the MN-CSE
- Setting of the *accessControlPolicyID* attribute

A straightforward approach to configure access control policies via an MN-AE for the considered example M2M service is outlined in the following steps:

1) List all resources which require assignment of access control policies. The resulting table of resources hosted by the MN-CSE for the given use case looks as follows:

| **resourceType** | **resourceName(s)** | **Description** |
|---|---|---|
| <CSEBase> | cb1 | CSEBase of MN-CSE |
| <AE> | adnae1<br>adnae2<br>mnae | <AE> of ADN-AE1<br><AE> of ADN-AE2<br><AE> of MN-AE |
| <container> | cnt1<br>cnt2 | door lock 1 container<br>door lock 2 container |
| <subscription> | subae1<br>subae2<br>subae1ae3 | subscription of ADN-AE1 to door lock 1<br>subscription of ADN-AE2 to door lock 2<br>subscription of ADN-AE3 to door lock 1 |

| | | |
|---|---|---|
| | subae2ae3 | subscription of ADN-AE3 to door lock 2 |

2) List all applicable entities (AEs and CSEs) and their identifiers (AE-IDs, CSE-IDs) from which request messages can originate and define which operations each entity is allowed to do. The entity identifiers will be included in the *accessControlOriginator* parameter. The identifier of the MN-AE is chosen to be "C-ACP-mgr". The identifiers of the other entities have been assigned already in the SAEF examples in clause 7.1. The applicable operations are specified with regard to the resources hosted by the MN-CSE in the table in step 1. The operations are represented as a string which indicates the allowed operations (C = Create, R = Retrieve, U = Update, D = Delete, N = Notify, d = discovery). the string "CRUDNd" means that all operations are allowed. The string "R" means that only Retrieve is allowed. The number in parenthesis is the encoded presentaion Since the door lock controller ADN-AE3 is registered to the IN-CSE, there bar access control policies related to this entity on the IN-CSE. These are out of scope of this example.

| **Entity** | **AE-ID or CSE-ID** | **Applicable operations** | **Comment** |
|---|---|---|---|
| ADN-AE1 | C-lock-AE1 | cb1: R (2)<br>adnae1: CRUDNd (63)<br>cnt1; CRUDNd (63)<br>sub1: CRUDNd (63) | Retrieve privilege on CSEBase<br>"default access privilege" on all resource created by itself, no access privilege to any other resource |
| ADN-AE2 | C-lock-AE2 | cb1: R (2)<br>adnae2: CRUDNd (63)<br>cnt2; CRUDNd (63)<br>sub2: CRUDNd (63) | Retrieve privilege on CSEBase<br>"default access privilege" on all resources created by itself, no access privilege to any other resource |
| ADN-AE3 | C-lockControl-AE3 | cb1: R (2)<br>adnae1: Rd (34)<br>cnt1: CRd (35)<br>adnae2: Rd (34)<br>cnt2; CRd (35)<br>subae1ae3: CRUDNd (63)<br>subae2ae3: CRUDNd (63) | Retrieve privilege on CSEBase<br>Retrieve and discover door lock <AE> resources<br>Create resources under door lock <container>, Retrieve and Discover <container><br>"default access privilege" on <subscription> resources created by itself |
| MN-AE | C-ACP-mgr | cb1: CRd (35)<br>mnae: CRUDNd (63)<br>adnae1: CRUDNd (63)<br>cnt1; CRUDNd (63)<br>adnae2: CRUDNd (63)<br>cnt2; CRUDNd (63)<br>sub1: CRUDNd (63)<br>sub2: CRUDNd (63)<br>subae1ae3: CRUDNd (63)<br>subae2ae3: CRUDNd (63) | Privilege to create children and Retrieve privilege on CSEBase<br>"default access privilege" on all resource created by itself (i.e. mnae and selfPrívileges of ACPs)<br>Privilege to perform all operations on all resource requiring access control |
| MN-CSE | mn-cse-123456 | cb1: CRUDNd (63) | all operations on <CSEBase> permitted, no other operations required for the present use case |

3) Convert entries of the table derived in step 2) into appropriate sets of access control rules.

Combine all entity IDs which are permitted to apply the same set of operations on the same resource into an access control rule (acr).

In the table below, acr's are represented in a pseudo JSON format, leaving away commas, braces and quotes around member names and values (see clause 8.5 of TS-0004 [i.3]).

| **Reference** | **accessControlRule** | **applicable to resource(s)** |
|---|---|---|
| acr1 | acor: [C-lock-AE1 C-lock-AE2 C-lockControl-AE3]<br>acop: 2 | cb1 |

| | | |
|---|---|---|
| acr2 | acor: [C-ACP-mgr]<br>acop: 35 | cb1 |
| acr3 | acor: [mn-cse-123456]<br>acop: 63 | cb1 |
| acr4 | acor: [C-lock-AE1 C-ACP-mgr]<br>acop: 63 | adnae1 |
| acr5 | acor: [C-lockControl-AE3]<br>acop: 34 | adnae1, adnae2 |
| acr6 | acor: [C-lock-AE2 C-ACP-mgr]<br>acop: 63 | adnae2 |
| acr67 | acor: [C-lock-AE1 C-ACP-mgr]<br>acop: 63 | cnt1, sub1 |
| acr8 | acor: [C-lock-AE2 C-ACP-mgr]<br>acop: 63 | cnt2, sub2 |
| acr9 | acor: [C-lockControl-AE3]<br>acop: 35 | cnt1, cnt2 |
| acr10 | acor: [C-lockControl-AE3 C-ACP-mgr]<br>acop: 63 | subae1ae3, subae2ae3 |
| acr11 | acor: [C-ACP-mgr]<br>acop: 63 | Mnae |

4) Merge multiple access control rules into suitable <accessControlPolicy> resources.

All access control rules which apply to the same resource can be combined into an individual <accessControlPolicy> resource. Each distinct set of accessControlRules defines a separate <accessControlPolicy> resource.

| **Resource** | **accessControlRules** | ***resourceName* of *<accessControlPolicy>*** |
|---|---|---|
| cb1 | acr1, acr2, acr3 | acp1 |
| adnae1 | acr4, acr5 | acp2 |
| adnae2 | acr6, acr6 | acp3 |
| mnae | acr11 | acp4 |
| cnt1 | acr7, acr8 | acp5 |
| cnt2 | acr8, acr9 | acp6 |
| sub1 | acr7 | acp7 |
| sub2 | acr8 | acp8 |
| subae1ae3 | acr10 | acp9 |
| subae2ae3 | acr10 | acp9 |

The *selfPrivileges* element of each <accessControlPolicy> resource is set to the default access privilege of the MN-AE, which is represented by acr10 in the table above.

5) Set the *accessControlPolicyIDs* attribute of the resources listed in the table of step 4) equal to the resource identifiers of <accessControlPolicy> resources acp1 ... acp9.

As a result of executing the above steps, all required access control policies are setup on the MN-CSE to operate the considered service in a fully oneM2M compliant way.

## 7.3 Secure communications

Once a security association is established between adjacent oneM2M nodes, all communication between these nodes is secured. However, all data of request and response messages is visible in the clear to both end points of a security association. Messages which need to be forwarded by an MN-CSE or IN-CSE are re-encrypted using the security context established with the next-hop node. Any intermediate CSE is trusted in this communication scenario. If a communication path includes CSEs which cannot be trusted, end-to-end security mechanisms need to be employed.

The present version of this document focuses on secure communication between adjacent nodes. Future versions will also address examples of configuring end-to-end communication using the ESPrim and ESData mechanisms specified in TS-0003 [i.4].

# *Proforma copyright release text block*

*This text box shall immediately follow after the heading of an element (i.e. clause or annex) containing a proforma or template which is intended to be copied by the user. Such an element shall always start on a new page.*



*<PAGE BREAK>*

# Annex A: Security Association Establishment Message Flows

## A.1 Introduction

This Annex presents some example message flows which are useful to understand the operation of the oneM2M security establishment frameworks, to verify correct operation or to identify the cause of misbehavior.

Some details of TLS message flows and message content depend on the employed SSL/TLS implementation. Implementations of oneM2M entities will typically make use of SSL/TLS libraries to enable support of the required security functions specified in TS-0003 [i.4]. Examples of open source SSL/TLS libraries include *OpenSSL*, *gnuSSL* and *mbed TLS*.

Such SSL/TLS libraries implement the basic cryptographic functions and provide various utility functions such as e.g. TLS clients and servers which may be executed from a command line.

The message flows shown here have been produced using OpenSSL Version 1.1.1-dev on an Ubuntu 14.04 computer using the s_client and s_server utility functions, and employing Wireshark for capturing and analyzing the exchanged data packets. Note that OpenSSL Version 1.1.0 or higher is required to support the PSK ciphers defined in RFC 5989 and mandated to be used by TS-0003 [i.4].

The commands given in the subsections below may be used to reproduce these flows.

## A.2 PSK-Based Security Association Establishment

A typical flow of messages and actions for a successful PSK-Based Security Association Establishment is shown in figure A.2-1. The message content described in the steps below applies to the example described in clause 7.1.2.

Subsequent to TCP connection establishment (not shown in the Figure), the following messages are exchanged between ADN-AE1 and the MN-CSE:

1. The TLS client on ADN-AE1 sends a Client Hello Handshake message which is encapsulated in a TLS Record layer frame. The record layer message includes the following fields:
   i. Record layer header fields:
      - Content type 0x16 (Handshake)
      - Version 0x0301 (indicating TLS 1.0)
      - Length of the message (2 bytes, value depending on the message content)
   ii. Application data (handshake message):
      - Handshake Type 0x01 (Client Hello)
      - Length of the message (3 bytes, value depending on the message content)
      - Client Version 0x0303 (TLS 1.2)
      - (Client) Random (32 bytes, generated by the TLS client's pseudo random number generator (PRNG))
      - Length of cipher suites field (value at least 1)
      - List of cipher suites supported by the client. It includes identifier for TLS_PSK_WITH_AES_128_CBC_SHA256 (0x00ae)
      - Extension length and Extensions (irrelevant for this example)
2. The TLS server handshake protocol responds with Server Hello and Server Hello Done messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.
   i. Record layer header fields:
      - Content type 0x16 (Handshake)
      - Version 0x0303 (indicating TLS 1.2)
      - Length of the application data field (2 bytes, value depending on the message content)
   ii. Application data ("Server Hello" handshake message):
      - Handshake Type 0x02 (Server Hello)
      - Length of the message (3 bytes, value depending on the message content)
      - Server version 0x0303 (indicating TLS 1.2)
      - (Server) Random (32 bytes, generated by the TLS server's PRNG)
      - Session-Id length (0x00, no session ID supplied)
      - Cipher suite selected by the server is TLS_PSK_WITH_AES_128_CBC_SHA256 (0x00ae)
      - Compression method (null, no compression)
      - Extension length and Extensions (irrelevant for this example)
   iii. Record layer header fields:
      - Same as in step 2.i
   iv. Application data ("Server Hello Done" handshake message):
      - Handshake type 0x0e (Server Hello Done)
      - Length of the message (0x0000, message has no content)
3. The TLS client responds with Client Key exchange, Change Cipher Spec, Finished messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.
   i. Record layer header fields:
      - Same as in step 2.i
   ii. Application data ("Client Key Exchange" handshake message):
      - Handshake Type 0x10 (Client Key Exchange)
      - Length of the message (3 bytes, value depending on the message content)
      - PSK client parameters:
         - Identity length ( 0x00000f in this example)
         - PSK Identity (here binary equivalent of "Client_identity")
   iii. Record layer header fields:
      - Content type 0x14 (Change Cipher Spec)
      - Version 0x0303 (TLS 1.2)
      - Length of the message (0x0001)
   iv. Application data ("Change Cipher Spec" message):
      - Change Cipher Spec message 0x01 (1 byte)

v. Record layer header fields:
   - Same as in step 2.i

vi. Application data (encrypted "Finished" handshake message)
   - Handshake type 0x14 (Finished)
   - Length of the message 0x00000c (12)
   - Verify Data (12 bytes), see RFC 5246, section 7.4.9.

4. The server retrieves Kpsa associated with the PSK Identity, computes the master secret and authenticates the client by validating Verify Data
5. The TLS server responds with New Session Ticket, Change Cipher Spec, Finished messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.

i. Record layer header fields:
   - Same as in step 2.i

ii. Application data ("New Session Ticket" handshake message):
   - Handshake Type 0x04 (New Session Ticket)
   - Length of the message (3 bytes: 0x0000b6)
   - Session Ticket:
     - Lifetime Hint (4 bytes: 0x00001c20, 7200 in this example)
     - Session Ticket Length (2 bytes, 0x00b0, 176 in this example)
     - Session Ticket (176 bytes), see RFC 4507, server session state enabling session resumption

iii. Record layer header fields:
   - Content Type 0x14 (Change Cipher Spec)
   - Version 0x0303 (TLS 1.2)
   - Length of the message (0x0001)

iv. Encrypted application data ("Change Cipher Spec" message):
   - Change Cipher Spec message 0x01 (1 byte)

v. Record layer header fields:
   - Same as in step 2.i

vi. Application data (encrypted "Finished" handshake message, to verify that the key exchange and authentication processes were successful):
   - Handshake Type 0x14 (Finished)
   - Length of the message 0x00000c (12)
   - Verify Data (12 bytes), see RFC 5246, section 7.4.9.

6. The client authenticates the server by validating Verify Data
7. Application data encrypted by the TLS record layer is exchanged between ADN-AE1 and MN-CSE

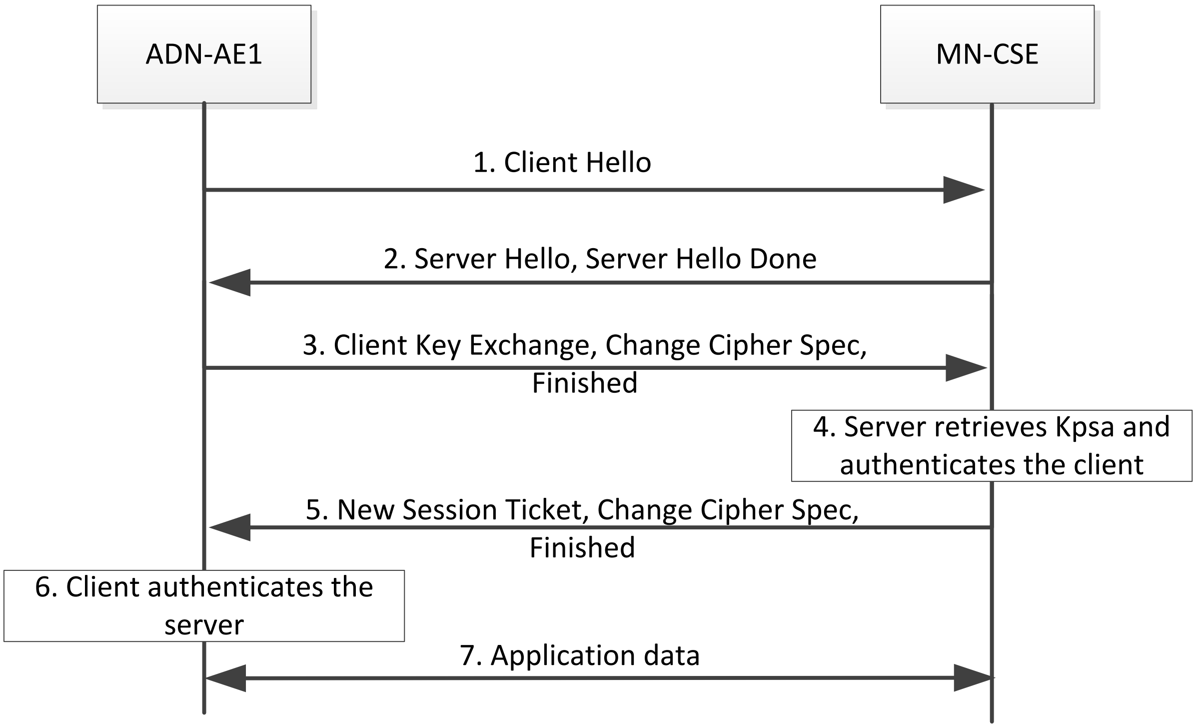


**Figure A.2-1: PSK-Based Security Association Establishment**

The message flow described above (excluding step 7) can be reproduced with the following commands under Linux OS using localhost IP address and port 443:

**TLS server on MN-CSE:**

```
$ sudo openssl s_server -accept 443 -psk 1a2b3c4d5e6f7a8b
```

**TLS Client on ADN-AE1:**

```
$ openssl s_client -connect 0.0.0.0:443 -psk_identity Client_identity \
                   -psk 1a2b3c4d5e6f7a8b -cipher PSK-AES128-CBC-SHA256
```

NOTE: The OpenSSL s_server utility does not support table lookup of pre-shared keys when using the option

```
-psk_identity AE123456789015-Lock@in.provider.com
```

as required for the example in clause 7.1.2. Therefore, the above command line for the server includes the used PSK itself. The client command line provides the PSK identity “Client_identity” which is expected by the server for this PSK.

Note that in order to enable Wireshark to decrypt application data which has been encrypted by the TLS record layer, it is configured as follows:

In the Wireshark configuration menu Edit -> Preferences -> Protocols -> SSL,

1) In the “Pre-Shared-Key” field, enter Kpsa, i.e. 1a2b3c4d5e6f7a8b
2) In the (Pre)-Master-Secret log filename field, enter the name of a text file which includes Client Random (32 bytes as 64 hex characters) and the Master Secret (48 bytes as 96 hex characters) as a text line as follows:

   CLIENT_RANDOM <space> 64-characters-random <space> 96-characters-Master-Secret

The master secret is provided as log information in the terminal window, where s_client is started. The value of Client Random can be retrieved from the Wireshark packet capture in the Client Hello handshake message.

First the data captured with Wireshark is stored into a file. Then, after configuring Wireshark as described above, the messages in the saved data file can be decrypted by Wireshark.

*Editor's note: relation between credential identifiers, entity identifiers and service subscription information needs to be clarified*

## A.3 Certificate-Based Security Association Establishment

Figure A.3-1 shows a typical flow of messages and actions for a successful certificate-based Security Association Establishment. The message content, i.e. the names of certificate files, private key files and CSE identifiers, described in the steps of the message flow, corresponds to the example described in clause 7.1.3.

Subsequent to TCP connection establishment (not shown in the Figure), the following messages are exchanged between ADN-AE1 and the MN-CSE:

1. The TLS client on MN-CSE sends a Client Hello Handshake message which is encapsulated in a TLS Record layer frame. The record layer message includes the following fields:
   i. Record layer header fields:
      - Content type 0x16 (Handshake)
      - Version 0x0301 (indicating TLS 1.0)
      - Length of the message (2 bytes, value depending on the message content)

   ii. Application data (handshake message):
      - Handshake Type 0x01 (Client Hello)
      - Length of the message (3 bytes, value depending on the message content)
      - Client Version 0x0303 (TLS 1.2)
      - (Client) Random (32 bytes, generated by the TLS client's pseudo random number generator (PRNG))
      - Length of cipher suites field
      - List of cipher suites supported by the client. This list includes TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
      - Extension length and Extensions (includes ec_point_formats, eliptic_curves, SessionTicket TLS, signature_algorithms)
2. The TLS server handshake protocol responds with Server Hello, Certificate, Server Key Exchange, Certificate Request and Server Hello Done messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.
   i. Record layer header fields:
      - Content type 0x16 (Handshake)
      - Version 0x0303 (indicating TLS 1.2)
      - Length of the application data field (2 bytes, value depending on the message content)

   ii. Application data ("Server Hello" handshake message):
      - Handshake Type 0x02 (Server Hello)
      - Length of the message (3 bytes, value depending on the message content)
      - Server version 0x0303 (indicating TLS 1.2)
      - (Server) Random (32 bytes, generated by the TLS server's PRNG)
      - Session-Id length (0x00, no session ID supplied)
      - Cipher suite selected by the server, should be TLS_PSK_WITH_AES_128_CBC_SHA256 (0x00ae)
      - Compression method (null, no compression)
      - Extension length and Extensions (only extension types included, irrelevant for this example)

   iii. Record layer header fields:
      - Same as in step 2.i

   iv. Application Data ("Certificate" handshake message): includes IN-CSE certificate and the Certificate
      - Handshake type 0x11 (Certificate)
      - Length of the message (3 bytes, value is 1224, for the given certificates)
      - Certificate length (3 bytes)
      - Certificate (601 bytes): MN-CSE certificate
      - Certificate length 3 bytes

- Certificate 614 bytes: IN-CSE certificate

v. Record layer header fields:
- Same as in step 2.i

vi. Application Data (“Server Key Exchange” handshake message):
- Handshake type 0x0c (Server Key Exchange)
- Length of the message (3 bytes)
- EC Diffie-Hellman Server Parameters

vii. Record layer header fields:
- Same as in step 2.i

viii. Application Data (“Certificate Request” handshake message):
- Handshake type 0x0d (Certificate Request)
- Length of the message (3 bytes)
- Certificate Types, Signature Hash Algorithms
- Distinguished Names, includes the issuer of the certificate

ix. Record layer header fields:
- Same as in step 2.i

x. Application data (“Server Hello Done” handshake message):
- Handshake type 0x0e (Server Hello Done)
- Length of the message (0x0000, message has no content)

3. The TLS client validates the certificate (chain) received from the TLS server.
   The client validates the signature(s) of the certificate(s) and checks if it can trust the root certificate.

4. The TLS client responds with Certificate, Client Key exchange, Certificate Verify, Change Cipher Spec, Finished messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.

i. Record layer header fields:
- Same as in step 2.i

ii. Application data (“Certificate” handshake message):
- Handshake Type 0x0b (Certificate)
- Length of the message (3 bytes, value depending on the message content, 608 bytes in this example)
- Certificates length (3 bytes, length of certificate chain, value is 605 bytes for the given certificate 02.pem)
- Certificate length (3 bytes, value is 602 bytes for the certificate given in 02.pem)
- Certificate (ASN.1 DER encoded binary representation of the certificate included in 02.pem)

iii. Record layer header fields:
- Same as in step 2.i

iv. Application data (“Client Key Exchange” handshake message):
- Handshake Type 0x10 (Client Key Exchange)
- Length of the message (3 bytes, value depending on the message content)
- PSK client parameters:
  - Identity length ( 0x00000f in this example)
  - PSK Identity (here binary equivalent of “Client_identity”)

vii. Record layer header fields:
- Same as in step 2.i

viii. Application data (“Certificate Verify” handshake message):
- Handshake Type 0x0f (Certificate Verify)
- Length of the message (3 bytes, value depending on the message content)
- Signature hash algorithm (ECDSA with SHA256, Signature Length (72 bytes) and Signature of all sent or received handshake messages of the current TLS handshake, see Section 7.4.8 of RFC5246

v. Record layer header fields:
- Same as in step 2.

vi. Application data (“Change Cipher Spec” message):
- Change Cipher Spec message 0x01 (1 byte)

vii. Record layer header fields:
- Same as in step 2.i

viii. Application data (encrypted “Finished” handshake message)
  - Handshake type 0x14 (Finished)
  - Length of the message 0x00000c (12)
  - Verify Data (12 bytes), see RFC 5246, section 7.4.9.

5. The server validates the certificate (chain) received from the client.
6. The TLS server responds with New Session Ticket, Change Cipher Spec, Finished messages. For the implementation employed here, each of these messages is encapsulated into a dedicated record layer frame.

i. Record layer header fields:
  - Same as in step 2.i

ii. Application data (“New Session Ticket” handshake message):
  - Handshake Type 0x04 (New Session Ticket)
  - Length of the message (3 bytes: 0x0000b6)
  - Session Ticket:
    - Lifetime Hint (4 bytes: 0x00001c20, 7200 in this example)
    - Session Ticket Length (2 bytes, 0x00b0, 176 in this example)
    - Session Ticket (176 bytes), see RFC 4507, server session state enabling session resumption

iii. Record layer header fields:
  - Content Type 0x14 (Change Cipher Spec)
  - Version 0x0303 (TLS 1.2)
  - Length of the message (0x0001)

iv. Encrypted application data (“Change Cipher Spec” message):
  - Change Cipher Spec message 0x01 (1 byte)

v. Record layer header fields:
  - Same as in step 2.i

vi. Application data (encrypted “Finished” handshake message, to verify that the key exchange and authentication processes were successful):
  - Handshake Type 0x14 (Finished)
  - Length of the message 0x00000c (12)
  - Verify Data (12 bytes), see RFC 5246, section 7.4.9.

7. The client authenticates the server by validating the Verify Data field and by matching of the CSE-ID in the subjectAltName field with its preconfigured registrar CSE-ID. Also, the server may check if the client’s MN-CSE-ID given in the subjectAltName field of the client certificate is already registered or is allowed to register to the IN-CSE (e.g. by checking if there is a <serviceSubscribedNode> resource instance which includes this MN-CSE ID.
8. Service Layer data encrypted by the TLS record layer is exchanged between MN-CSE and IN-CSE

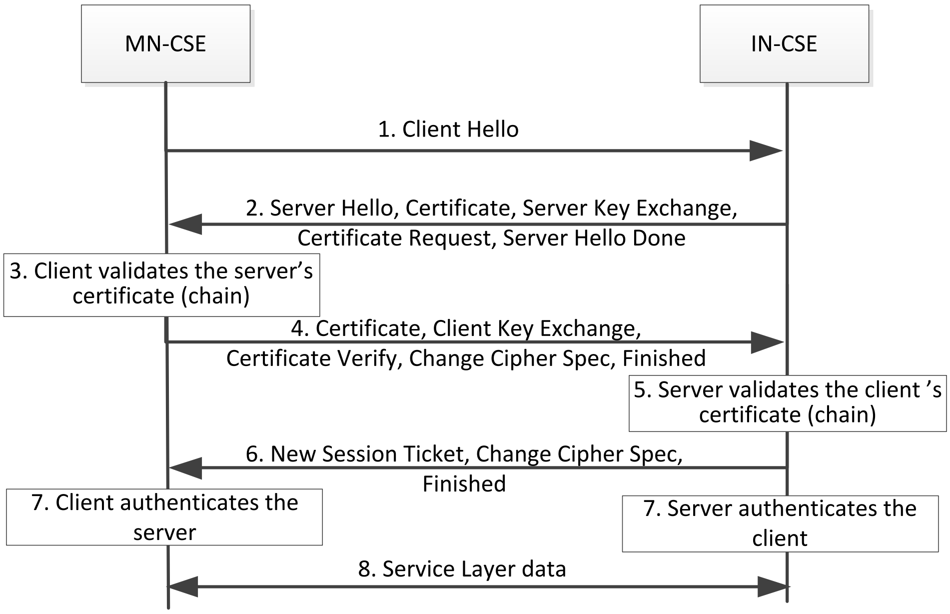


**Figure A.3-1: Certificate-Based Security Association Establishment**

The message flow described above (excluding step 7) can be reproduced with the following commands under Linux OS using localhost IP address and port 443 (it is assumed that path names apply and CSE-certificates are available in the directory from where this command is issued):

**TLS server on IN-CSE:**

```
$ sudo openssl s_server -accept 443 -Verify 1 -key in_cse_key.pem \
               -cert 01.pem -CApath ./demoCA -CAfile ./demoCA/cacert.pem
```

**TLS client on MN-CSE:**

```
$ openssl s_client -connect 0.0.0.0:443 -key mn_cse_key.pem -cert 02.pem \
                   -verify 1 -cipher ECDHE-ECDSA-AES128-SHA256 \
                   -CApath ./demoCA -CAfile ./demoCA/cacert.pem
```

NOTE: CipherSuite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 = {0xC0,0x23} as defined in RFC5989 is denoted ECDHE-ECDSA-AES128-SHA256 in openssl [i.9]

Note that in order to enable Wireshark to decrypt application data which has been encrypted by the TLS record layer, it is configured as follows:

In the Wireshark configuration menu Edit -> Preferences -> Protocols -> SSL,

- In the (Pre)-Master-Secret log filename field, enter the name of a text file which includes Client Random (32 bytes as 64 hex characters) and the Master Secret (48 bytes as 96 hex characters) as a text line as follows:

    CLIENT_RANDOM <space> 64-characters-random <space> 96-characters-Master-Secret

The master secret is provided as log information in the terminal window, where s_client is started. The value of Client Random (comprised of GMT Time (4 bytes/8 hex chars) plus Random (28 bytes/56 hex chars)) can be retrieved from the Wireshark packet capture in the Client Hello handshake message.

# A.4 MAF-Based Security Association Establishment

In MAF-based Security Association Establishment between two oneM2M entities (i.e. AEs and CSEs) symmetric key credentials are employed which have been established with a preceding procedure on a MAF. This key establishment procedure corresponds to steps 1 to 6 in the example described in clause 7.1.4.

Step 1 of the procedure in clause 7.1.4 represents a certificate-based TLS-handshake between MAF client and MAF where in addition the keying material exporter function as defined in RFC 5705 (RFC 65705) is enabled.

The handshake message flow of this step can be produced with the following commands under Linux OS using a DNS-resolvable MAF-FQDN *myMAF.provider.org* and port 443 (it is assumed that path names apply and certificates are available in the directory from where this command is issued):

**TLS server on MAF with example FQDN *myMAF.provider.org*:**

```
$ sudo openssl s_server -accept 443 -Verify 1 -key maf_key.pem \
               -cert maf_cert.pem -CApath ./demoCA -CAfile ./demoCA/cacert.pem \
               -keymatexport EXPORTER-oneM2M-Connection -keymatexportlen 48
```

**TLS client on MAF client associated with AE3:**

```
$ openssl s_client -connect myMAF.provider.org:443 -key maf_client_key.pem  \
                   -cert maf_client_cert.pem -verify 1 –cipher ECDHE-ECDSA-AES128-SHA256\
                   -keymatexport EXPORTER-oneM2M-Connection -keymatexportlen 48
```

At both TLS endpoints, openssl produces an output such as the following (example):

```
Keying material exporter:
  Label: 'EXPORTER-oneM2M-Connection'
    Length: 48 bytes
    Keying material: FF15D84E3E38D6974B0EB3E5606C85FE
                     37F61D5A7FEA1E9CFD8DB76D2F8B6230
                     130EF8A84F9F9F967DA385867984EED0
```

The value of `Keying material` is a 48 byte array represented as a 96-character hexadecimal string which is divided into two parts:

- upper 16 bytes (32 hex characters), denoted as Connection Key Identifier (KcID):
    - FF15D84E3E38D6974B0EB3E5606C85FE
- lower 32 bytes (64 hex characters), denoted as M2M Secure Connection Key (Kc):
    - 37F61D5A7FEA1E9CFD8DB76D2F8B6230130EF8A84F9F9F967DA385867984EED0

From KcID, the *Key Identifier* is derived as follows (see clause 10.3.5 of TS-0003 [i.4]):

```
Key Identifier = RelativeKeyID@MAF-FQDN
```

where RelativeKeyID = hexBinary(KcID) and MAF-FQDN is the domain name of the MAF on which the key Kc which is associated with the Key Identifier is registered. For the above example of MAF-FQDN and KcID, the Key Identifier is derived as:

```
hexBinary(0xFF15D84E3E38D6974B0EB3E5606C85FE) = 'FF15D84E3E38D6974B0EB3E5606C85FE'

Key Identifier:  'FF15D84E3E38D6974B0EB3E5606C85FE@myMAF.provider.org'
```

Note that the value of the *resourceID* attribute of <*symmKeyReg*> resources instances hosted on the MAF identified by MAF-FQDN is set to the RelativeKeyID.

The hexadecimal representation of the key Kc associated with this Key Identifier will be stored in the *keyValue* attribute of a <*symmKeyReg*> resource instance, which is created in step 4 of the message sequence given in Figure 7.1.4-1.

# Annex B: Generation of Certificates

## B.1 Introduction

This Annex describes how to generate certificates which are compliant with the requirements defined in TS-0003 [i.4].

Generation of certificates requires setting up a simple Public Key Infrastructure (PKI). It is outlined here how this can be accomplished using OpenSSL. For simplicity a root CA is setup which employs a self-signed root certificate to sign all end user's certificates. The end users of the certificates in the present context refer to AEs or CSEs.

The private keys and certificates need to be deployed in AEs and CSEs in a secure way. Private keys require special protection on devices. They should be stored and be employed for security procedures in a secure environment. Note that these aspects are not addressed in this Annex. A simple way to protect keys is to store them in password protected files. However, for simplicity, in the following procedures this feature is not used.

Furthermore, the following conditions and conventions apply:

- all generated keys support elliptic curve Diffie-Hellman encryption (ECDHE) and elliptic curve digital signature Algorithm (ECDSA),
- all keys and certificates are generated in Privacy-Enhanced Mail (PEM) format and are stored in files with extension *.pem*,
- the described examples have been tested using OpenSSL v1.1.1-dev under a Ubuntu 14.04 LTS operating system.

Note that any addresses used in the examples shown in the present annex, e.g. in the issuer and subject fields of the generated certificates, are just arbitrary examples not applicable for real implementations.

## B.2 Setting up a root CA

When installing OpenSSL on a Linux computer, a configuration file openssl.cnf is created by default in the directory /etc/ssl.

The information in openssl.cnf defines sets of parameters which are applied by default by the openssl command line utility functions. Additional information on OpenSSL PKI and certificate generation can be found in [i.7] and [i.8].

The following section should be included into the default version of openssl.cnf to get the commands shown below and in clause B.3 to work properly:

```
####################################################################
[ ca ]
default_ca = CA_default                # The default ca section

####################################################################
[ CA_default ]

dir = ./demoCA                         # Where everything is kept
certs = $dir/certs                     # Where the issued certs are kept
crl_dir = $dir/crl                     # Where the issued crl are kept
database = $dir/index.txt              # database index file.
```

```
unique_subject = no                    # Set to 'no' to allow creation of
                                       # several certificates with same subject.
new_certs_dir = $dir/newcerts          # default place for new certs.
certificate  = $dir/cacert.pem         # The CA certificate
serial = $dir/serial                   # The current serial number
crlnumber  = $dir/crlnumber            # the current crl number
                                       # must be commented out to leave a V1 CRL
crl = $dir/crl.pem                     # The current CRL
private_key = $dir/private/cakey.pem   # private key of the root cert

RANDFILE = $dir/private/.rand          # private random number file
                                       # (not used in the present example)
x509_extensions = usr_cert             # The extensions to add to the cert

[signing_policy]
countryName            = optional
stateOrProvinceName    = optional
localityName           = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional
subjectAltName         = supplied
```

Create or change to some existing directory, where the tree containing private keys and certificates should originate. From this directory, execute the following commands:

```
$ mkdir demoCA
$ mkdir demoCA/newcerts
$ mkdir demoCA/private
$ sh -c "echo '01' > ./demoCA/serial"
$ touch ./demoCA/index.txt
```

These commands create the directory structure and the files which control the generation of the serial number of the certificates. The serial number of the end user certificates created by the CA will be incremented starting from 01.

## B.3 Generation of CA private key and root certificate

The command given below generates a CA key in a file cakey.pem with implicit elliptic curve parameters from the curve named secp256r1 (note that OpenSSL uses curve prime256v1 which is the same as secp256r1):

```
$ openssl ecparam -name secp256r1 -genkey -out cakey.pem
```

The command below generates a self-signed root certificate with the name cacert.pem:

```
$ openssl req -new -x509 -extensions v3_ca -key cakey.pem -subj
"/C=US/ST=California/O=Trusted Certificate
Authority/CN=mtrusted_ca.com/emailAddress=service@trusted_ca.com" -out cacert.pem -days
3650
```

The private key and certificate files need be moved into the directories as configured in openssl.cnf:

```
$ mv cakey.pem demoCA/private/.
```

```
$ mv cacert.pem demoCA/.
```

## B.4 Generation of end user private key and certificates

This clause shows commands which generate the end user certificates which are signed by the root CA. These certificates are employed in the example described in Annex A.3 by the IN-CSE and MN-CSE. The Subject Alternative Name of these certificates include the CSE-IDs of the IN-CSE and MN-CSE, respectively.

The following commands generate the key files:

```
$ openssl ecparam -name secp256r1 -genkey -out in_cse_key.pem
```

```
$ openssl ecparam -name secp256r1 -genkey -out mn_cse_key.pem
```

The following commands generate signing requests (CSRs) for the IN-CSE and MN-CSE certificates:

```
$ openssl req -new -extensions SAN -key in_cse_key.pem -subj
"/C=US/ST=California/O=MY_M2M_PROVIDER, Inc./CN=my.m2mprovider.org" -reqexts SAN -config
<(cat /etc/ssl/openssl.cnf <(printf "[SAN]\nsubjectAltName=DNS:my.m2mprovider.org/in-
cse")) -out in_cse_cert.csr -days 365
```

```
$ openssl req -new -extensions SAN -key mn_cse_key.pem -subj
"/C=US/ST=California/O=MY_M2M_PROVIDER, Inc./CN=my.m2mprovider.org"
-reqexts SAN -config <(cat /etc/ssl/openssl.cnf <(printf
"[SAN]\nsubjectAltName=DNS:my.m2mprovider.org/mn-cse")) -out mn_cse_cert.csr -days 365
```

The following command generate the signed IN-CSE certificate from the CSR. This produces a certificate `./demoCA/newcerts/01.pem`:

```
$ openssl ca -in in_cse_cert.csr -policy signing_policy -config /etc/ssl/openssl.cnf -
extensions SAN -config <(cat /etc/ssl/openssl.cnf <(printf
"[SAN]\nsubjectAltName=DNS:my.m2mprovider.org/in-cse")) -verbose
```

The following command generate the signed MN-CSE certificate from the CSR. This produces a certificate `./demoCA/newcerts/02.pem`:

```
$ openssl ca -in mn_cse_cert.csr -policy signing_policy -config /etc/ssl/openssl.cnf
-extensions SAN -config <(cat /etc/ssl/openssl.cnf <(printf
"[SAN]\nsubjectAltName=DNS:my.m2mprovider.org/mn-cse-123456")) -verbose
```

The private keys and certificates would need to be deployed on the end entities (i.e. IN-CSE with CSE-ID = in-cse and MN-CSE with CSE-ID = mn-cse-123456).

For testing of certificate-based TLS-handshake as described in Annex A.3,  these certificates and private keys may be copied into the directory from where the opennssl s_server and s_client commands given in Annex A.3 are executed.

# History

*This clause shall be the last one in the document and list the main phases (all additional information will be removed at the publication stage).*

| **Publication history** | | |
|---|---|---|
| V.1.1.1 | <dd Mmm yyyy> | <Milestone> |
| | | |
| | | |
| | | |
| | | |

| **Draft history** (to be removed on publication) | | |
|---|---|---|
| V.0.0.1 | 05 December 2016 | Initial skeleton |
| V0.1.0 | 21 February 2017 | Integration of contributions agreed during TP 27:<br>SEC-2017-0009R02<br>SEC-2017-0020R02<br>SEC-2017-0021R02 |
| V0.2.0 | 05 April 2017 | Integration of contributions agreed during TP 28:<br>TST-2017-0097R01 |
| V0.2.1 | 09 October 2017 | Integration of contributions agreed during TP 28:<br>SEC-2017-0138R01 |
| V0.3.0 | 04 December 2017 | Integration of contributions agreed during TP 32:<br>TST-2017-0259R02 |
| V0.4.0 | 30 January 2018 | Integration of contributions agreed during TP 33:<br>TST-2018-0010R03 |
| V0.5.0 | 23 March 2018 | Integration of contributions agreed during TP 34:<br>TST-2018-0038R03-TR-0038 |