IEC/IEEE 60802 Security Slice

Contributors
Fischer, Kai <kai.fischer@siemens.com>
Furch, Andreas <andreas.furch@siemens.com>
Pfaff, Oliver <oliver.pfaff@siemens.com>
Pössler, Thomas <thomas.poessler@siemens.com>
Steindl, Günter <guenter.steindl@siemens.com>

Abstract

The purpose of this text is to establish a common understanding of TSN-IA security. An incremental procedure is applied in bottom-up style:

i. First increment (V0.1 and V0.2, *prior versions*): establishing TLS with IA components (in TLS server role) that boot with factory defaults; provides chapters 1 to 4.1

ii. Second increment (V0.3, *prior version*): equipping IA components with trust anchors and credentials for NETCONF-over-TLS; provides chapter 4.2

iii. Third increment (V0.4, ***this version***): securely using IA components with NETCONF/YANG exchanges; provides chapter 5 and Annex D

iv. Forth increment (V0.5, *later*): equipping IA components with trust anchors and credentials for other exchanges (non-NETCONF/YANG); will provide chapter 6

v. Fifth increment (V0.6, *later*): securely using IA components with other exchanges (non-NETCONF/YANG); will provide chapter 7

Elaborations of this text provide a skeleton for the security profile text in D1.3 of TSN Profile for Industrial Automation. It also provides a background for describing the security use cases.

Log

v0.1	2021-05-21	Initial draft
v0.2	2021-06-11	Editorial changes, document structure refined, elaboration on the bootstrapping challenge (chapter 4.1) and corresponding sequence charts (Annex C)
v0.3	2021-06-25	Elaboration on the imprinting challenge (chapter 4.2)
v0.4	2021-07-09	Resource access authorization and message exchange protection and for NETCONF-over-TLS (chapter 5, Annex D)

Contents

References

[1] IETF RFC 4949: Internet Security Glossary, Version 2, 2007

[2] IETF RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, 2008

[3] IETF RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008

[4] IETF RFC 5890: Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework, 2010

[5] IETF RFC 5891: Internationalized Domain Names in Applications (IDNA): Protocol, 2010

[6] IETF RFC 6125: Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS), 2011

[7] IETF RFC 6241: Network Configuration Protocol (NETCONF), 2011

[8] IETF RFC 7589: Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication, 2015

[9] IETF RFC 7950: The YANG 1.1 Data Modeling Language, 2016

[10] IEEE 802.1AR-2018: IEEE Standard for Local and Metropolitan Area Networks–Secure Device Identity, 2018

[11] IETF RFC 8341: Network Configuration Access Control Model, 2018

[12] IETF RFC 8342: Network Management Datastore Architecture (NMDA), 2018

[13] IETF RFC 8366: A Voucher Artifact for Bootstrapping Protocols, 2018

[14] IETF RFC 8572: Secure Zero Touch Imprinting (SZTP), 2019

[15] IETF RFC 8995: Bootstrapping Remote Secure Key Infrastructure (BRSKI), 2021

[16] IETF NETCONF WG: A YANG Data Model for a Truststore (draft-ietf-netconf-trust-anchors-15), Internet Draft, Work in Progress, 2021

[17] IETF NETCONF WG: A YANG Data Model for a Keystore (draft-ietf-netconf-keystore-22.html), Internet Draft, Work in Progress, 2021

[18] IETF NETCONF WG: YANG Data Types and Groupings for Cryptography (draft-ietf-netconf-crypto-types-20.html), Internet Draft, Work in Progress, 2021

Abbreviations

AEAD	Authenticated Encryption with Added Data
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
ASN	Abstract Syntax Notation
CA	Certification Authority
CBC	Cipher Block Chaining
CMS	Cryptographic Message Syntax
CN	Common Name (X.500)
CRL	Certificate Revocation List
CRUDX	Create Read Update Delete eXecute
CSR	Certificate Signing Request
DAC	Discretionally Access Control
DER	Distinguished Encoding Rules
DH	Diffie-Hellman
DHE	Diffie-Hellman Ephemeral
DN	Distinguished Name (X.500)
DNS	Domain Name Service
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
EE	End Entity
GCM	Galois Counter Mode
HMAC	Keyed-Hashing for Message Authentication
FQDN	Fully Qualified Domain Name
HW	HardWare
IA	Industrial Automation
IA-ME	Industrial Automation Management Entity
IDevID	Initial Device IDentifier
LDevID	Locally significant Device Identifier
MAC	Message Authentication Code or Mandatory Access Control (security)
	Media Access Control (networking)
NACM	Network configuration Access Control Model
NETCONF	NETwork CONFiguration
NMDA	Network Management Datastore Architecture
OCSP	Online Certificate Status Protocol

OoB	Out-of-Band
PEM	Privacy Enhanced Mail
PFS	Perfect Forward Secrecy
PII	Personally Identifiable Information
PKCS	Public Key Cryptography Standards
RBAC	Role-Based Access Control
RSA	Rivest Shamir Adleman
SAN (or san)	Subject Alternative Name
SHA	Secure Hash Algorithm
SZTP	Secure Zero Touch Provisioning
TDME	TSN Domain Management Entity
TLS	Transport Layer Security
TOFU	Trust On First Use
TTP	Trusted Third Party
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
WG	Working Group
YANG	Yet Another Next Generation

1 Preconditions

Following preconditions are assumed:

- IA systems are equipped with system components from multiple manufacturers.
- Each individual system component has a housing that carries an end station or bridge component.
- By the time a system component is shipped by its manufacturer, it is assumed to comprise the following as part of its factory defaults:
    - **IDevID credential** object: defined by IEEE 802.1AR, see [10], to be further profiled by IEC/IEEE 60802. This object encompasses[1]:
        - Private key
        - End entity (EE) certificate (plus intermediate CA certificates) containing **product master data** identifying the physical instance of this component according to manufacturer knowledge e.g., product serial number and in an eternal manner.

      Note: IDevID EE certificates cannot contain deployment master data e.g., application name(s) or IP address(es).
    - Corresponding **trust anchor**: also defined by IEEE 802.1AR, see [10]. This object represents the manufacturer certification authority (CA), often in the form of a self-signed CA certificate. It is used to initialize the validation of certification paths of peers, see [3].
    - **Secure element** component: generic or dedicated HW (the exact form factor is out-of-scope for IEC/IEEE 60802) providing:
        - Persistent storage for keys and credentials esp. IDevID/LDevID credentials and corresponding trust anchors (see below)
        - Execution environment for these keys and credential

      Note: this is also known as **DevID module** in IEEE 802.1AR, see [10]

[1] Hint: *IDevID EE certificates can be thought of as "birth certificates" - they contain data that is known by the time-of-birth.*

- System components that are deployed in a production cell/site are equipped with an IP address.

2 Goal

A system component (that fulfills the prerequisites above) shall participate in protected network configuration. Assumptions:

- Network configuration uses NETCONF/YANG according [7] and [9]
- Secure transport for NETCONF is TLS according [8]
- The system component acts in (NETCONF and TLS) server role – its network configuration happens according to a push supply

Using NETCONF-over-TLS is straightforward provided the NETCONF-over-TLS server (i.e., the to-be-managed system component) possesses:

- A credential that matches the requirements in sections 6 of RFCs 7589 (see [8]) resp. RFC 6125 (see [6]): the component's FQDN has to be part of the `subjectAltName` extension in its EE certificate
- Trust anchor(s) that allow to validate the EE certificates (plus intermediate CA certificates) of its NETCONF-over-TLS clients.

Important: these objects are not available when the to-be-managed system component boots with its factory defaults. This text addresses this challenge as follows:

- Chapters 3 and 4 describe the equipment of IA components with credentials and trust anchors required for NETCONF-over-TLS. This applies resp. happens when IA components boot with factory defaults.
- Chapter 5 describes the secure management of IA components with NETCONF/YANG using TLS as secure transport. This applies resp. happens after IA components were equipped with credentials and trust anchors for NETCONF-over-TLS (explained in chapters 3 and 4).
- Chapters 6 describes the equipment of IA components with credentials and trust anchors required for other exchanges than NETCONF-over-TLS. This applies resp. happens after IA components were equipped with credentials and trust anchors for NETCONF-over-TLS (explained in chapters 3 and 4).
- Chapter 7 describes the secure employment of IA components in other exchanges than NETCONF/YANG. This applies resp. happens after IA components were equipped with credentials and trust anchors for other exchanges than NETCONF-over-TLS (explained in chapter 6).

3 Identifying the Challenges

3.1 Imprinting Challenge

Supply the **LDevID-NETCONF** credential and corresponding **trust anchor** in a secure manner to a system component that is booting from factory default state[2] and that shall be managed by means of NETCONF-over-TLS. Notes:

[2] The imprinting of an IA component with its LDevID-NETCONF credential as well as the corresponding trust anchor shall happen once when booting from factory default state.

- The shorthand term LDevID-NETCONF is used for an LDevID[3] credential according to IEEE 802.1AR (see [10]) which also matches the requirements that are set forth in sections 6 of RFC 7589 (see [8]) resp. RFC 6125 (see [6]).
- The specific term ‘imprinting’ is used for equipping IA components with the LDevID-NETCONF credential and corresponding trust anchor instead of the generic term ‘provisioning’ (can refer to any supply, is not limited to credentials and trust anchors)

Suggested approach for solving this imprinting challenge[4]: use NETCONF-over-TLS for supplying the LDevID-NETCONF credential and corresponding trust anchor. The LDevID-NETCONF credential and corresponding trust anchor supply happens in NETCONF payload according to a YANG model.

3.2 Bootstrapping Challenge

When this imprinting happens the to-be-provisioned objects cannot be simultaneously used in the TLS layer[5]. Other credentials and trust anchors must be used in the TLS layer when performing NETCONF-over-TLS exchanges for imprinting the LDevID-NETCONF credential and corresponding trust anchor.

Suggested approach for solving this bootstrapping challenge: use the IDevID credential and corresponding trust anchor on TLS level when doing the NETCONF-over-TLS exchanges to provision the LDevID-NETCONF credential and corresponding trust anchor.

This approach results in several sub-challenges that are identified below.

3.2.1 Server Identity Checking Challenge

As a client that is performing this imprinting, how to check the server identity before supplying sensitive resources to it (the LDevID-NETCONF credential)?

Note: the RFC 7589 (see [8]) resp. RFC 6125 (see [6]) matching rule is geared towards server identity checking in a post imprinting phase (“*all is setup*”). When RFC 7589 resp. RFC 6125 matching would be used during the credential imprinting phase, it would prohibit the supply.

3.2.2 Client Identity Verification Challenge

As a to-be-provisioned server (the IA component), how to check the client identity before accepting critical changes of the own state (the trust anchor that allows to validate the LDevID-NETCONF and other EE certificates presented by peer entities)?

Note: clients that call the IA component for doing the imprinting must be assumed to be equipped with credentials from an authority that is not yet known by the to-be-provisioned IA component which is booting from factory default.[6]

---

[3] In general, LDevID credentials encompass:

- Private key
- EE certificate containing **deployment master data** identifying the component according to deployment knowledge e.g., application name(s) or IP address(es) and in a time-limited manner.

Hint: *LDevID EE certificates can be thought of as “driving licenses” - they contain info that is unknown when “birth certificates” are issued e.g., driving license classes*

[4] NETCONF SZTP in [14] is no (full) solution for this imprinting challenge: it does not cover the credential portion. The trust anchor portion is covered but SZTP uses pull or physical push (*Removeable Storage*)

[5] The TLS handshake that demands the objects happens before the NETCONF application exchange.

[6] Albeit RFC 5246 is not explicit on what must happen when certification path validation fails, it is fair to expect the vast majority of server-side implementations to interrupt a TLS handshake when seeing a client certificate that cannot be validated with the already configured trust anchors.

3.2.3 Client Authorization Challenge

As a to-be-provisioned server (the IA component), how to determine whether the current client is authorized[7] to perform the imprinting of LDevID-NETCONF credential and trust anchor?

Note: RFC 8341 (NACM, see [11]) is geared towards authorizing operations in the post imprinting phase ("*all is setup*"). When RFC 8341 authorization would be used during the credential and trust anchor imprinting phase, it would prohibit this supply.

4 Solving the Challenges

4.1 Bootstrapping Challenge

Using the mechanisms described below, the bootstrapping part of the imprinting challenge can be solved.

4.1.1 Server Identity Checking Challenge

The IA component exposes a NETCONF service over TLS that is using its IDevID credential for authenticating itself while booting from factory default state and to be imprinted with an LDevID-NETCONF credential.

This provides following actuals to the imprinting client for checking the server:

- The `issuer` field in the IDevID EE certificate. IEEE 802.1AR (see [10]) requires this value to present a domain of uniqueness for the product serial number.
- The product serial number value from the IDevID EE certificate. IEEE 802.1AR requires this value to be provided in a `serialNumber` attribute[8] of the `subject` field.

Before imprinting the LDevID-NETCONF credential, the imprinting client checks the actual server identity that is stated by the IA component on TLS level by matching against:

- A list of accepted (or blocked) manufacturers

  Note: matching between legal registration or common names on root level[9] and X.500 name on leaf level[10] representations. The caveat is: X.500 issuer names are mandated for X.509 certificates but uncommon outside the PKI domain. TODO: discussion is needed if a matching shall be specified in TSN-IA (normative text) or whether TSN-IA just provides some background (informative text).

- Per accepted manufacturer, a list of accepted (or blocked) product instances by their product serial number incl. wildcards

Details of how this matching happens depends on the implementation of the client that performs this imprinting. For example:

- A human-operated imprinting client might trigger a dialogue by displaying the actuals and asking for an "Okay or not okay?" input by its operator before proceeding. The operator then performs this checking OoB - from the perspective of the client.
- An automatedly operating imprinting client might demand to be (pre-)configured with input about the "expected" system components and performs an automated checking.

———————————

[7] There is also a post-imprinting client authorization challenge (not considered here): as an already provisioned server, how to determine whether a client is authorized to perform its network configuration actions?

[8] This attribute is identified by the OID 2.5.4.5 which is defined by X.520 (see RFC 4519).

[9] E.g. "Antarctica; Super-Duper-Manufacturer, Inc.; Place of Registration: McMurdo, AQ; Registered Office Address: 77, Mt. Erebus Drive, McMurdo, AQ; Registration Ref.: XY-4711"

[10] E.g. "C=AQ,O=Super-Duper-Manufacturer,OU=Industrial Automation,CN=IDevID Issuing CA V1.0"

Items to follow-up in a discussion with IEEE Security WG (regarded a TODO): Home of product serial number (subject name (as serial number attribute) vs. subject alternative name). Consideration of industry-wide unique product instance identifiers in addition (or instead) to the current product instance identifiers that are (at most) manufacturer-wide unique

4.1.2 Client Identity Verification Challenge

The IA component exposes a NETCONF service over TLS that is using its manufacturer installed trust anchors for authenticating clients while booting from factory default state and to be imprinted with a trust anchor (that allows to validate LDevID-NETCONF and other EE certificates presented by peer entities).

This (and only this) endpoint performs a “provisional accept of client cert”[11] according following procedure:

1. Challenge the client for TLS client authentication (required by RFC 7589, see [8]) by sending a `CertificateRequest` message (required by RFC 5246, see [2]) with an empty `certificate_authorities` entry
2. Perform certification path validation according to RFC 5280 (see [3]) for the contents of the client’s `Certificate` message (fail if the certificate list in this message is empty)
3. Provisionally accept a failing certification path validation when the reason is ‘no matching trust anchor’ (and only this reason) and proceed with the TLS exchanges.
4. Expect the client to send a trust anchor in the NETCONF application payload over this provisionally accepted TLS session (nothing else). This shall happen in one of two forms (see chapter 4.2 for further details of this supply):
    a. *Plain form*: a raw X.509 CA certificate as part of a YANG object. Only syntax and simple hygiene checks are possible in this case, no actual cryptographic checks. This object is accepted when syntax and hygiene checks are passed. This provides a TOFU model.
    b. *Protected form*: an X.509 CA certificate that is embedded in a voucher (RFC 8366, see [13]) as part of a YANG object. The voucher is a signed object that can be cryptographically checked with the manufacturer-provided trust anchors. This object is accepted when cryptographic as well as syntax and hygiene checks are passed.

       TODO: elaborate on delegation models, voucher object flavors/details (with/without nonce etc)
5. If the trust anchor in the NETCONF application payload was accepted, then redo the certification path validation using this object (see step 2).
6. If this revalidation is successful, then the client identity is successfully established.
7. If client identity is established, perform the client authorization (see below):
    a. If authorized: persist the provisioned trust anchor and use it for subsequent certification path validation operations
    b. Else: refuse the supplied trust anchor

[11] This is a mirrored version of the “provisional accept of server cert” in RFC 8995 (see [15])

4.1.3 Client Authorization Challenge

The authorization of clients for the task of imprinting the LDevID-NETCONF credential and the corresponding trust anchor when booting from factory default state is subject to the security model for imprinting the trust anchor:

- *Plain form*: in the TOFU case, the to-be-provisioned server (the IA component) has no reasonable means to distinguish the following cases:
    - Client is authenticated and authorized for doing this imprinting
    - Client is authenticated but not authorized for doing this imprinting

  Hence in the TOFU model all authenticated clients are accepted as authorized for doing the imprinting of the LDevID-NETCONF credential and the corresponding trust anchor. Only contextual checks such as "once only when bootstrapping from factory default" (first-one-wins) are feasible. TODO: discuss whether such contextual checks shall be described in a normative way

- *Protected form*: in the voucher case, the details of an authorization model are up to the manufacturer as voucher object production is done (or delegated) by the manufacturer and voucher object consumption is done by a product of this manufacturer. This allows to support various models including:
    - Any client of any owner/operator organization can perform this imprinting – voucher is not bound to owner/operator organization and/or their clients
    - Any client of a dedicated owner/operator organization can perform this imprinting – voucher is bound to an owner/operator but not to their clients
    - Only dedicated clients of a dedicated owner/operator organization can perform this imprinting – voucher is bound to an owner/operator organization as well as to dedicated clients

  Detailing such bindings is out-of-scope for IEC/IEEE 60802.

4.2 Imprinting Challenge

4.2.1 Use Cases

- `imprintTrustAnchor`: imprint a local, deployment-specific trust anchor[12] (LDevID) to an IA component that is booting with factory defaults. Subcases:
    - Trust anchor is provided in plain form[13] (TOFU) e.g., a X.509 certificate in enveloped form without protection (such as: degenerated CMS SignedData, "certs-only" [no signature], RFC 5652) or in raw form (ASN.1 DER binary, opt. Base64-encoded and wrapped with PEM markers)
    - Trust anchor is provided in protected form[14] e.g., a X.509 certificate in enveloped form with protection (such as: CMS SignedData [not degenerated] or a voucher object [RFC 8366])
- `imprintCredential`: imprint a local, deployment-specific credential[15] (LDevID) to an IA component that is booting with factory defaults. Subcases:

[12] An X.509 CA certificate that is used as an input for certification path validation (see section 6 of RFC 5280)

[13] The verification of a self-signed root CA certificate only provides the integrity of this object, not its authenticity. In other words: anybody can issue a self-signed root CA certificate object for which the signature validation works, that appears to represent e.g., the United Nations but where its private key is controlled by another entity.

[14] To establish authenticity for self-signed root CA certificate additional means are needed. Embedding self-signed root CA certificates into RFC 8366 voucher objects provides one means to establish that.

[15] A private key and the corresponding X.509 EE certificate, optionally plus intermediate sub-CA certificates

- IA component-external key generation
- IA component-internal key generation

TODO: `imprintUsernames`, `imprintUserRules`, see figures in sections C.1 (required objects) vs. C.2 (available objects when booting with factory defaults); deferred from V0.3 for complexity reasons (`imprintTrustAnchor/imprintCredential` occupy ca. 10 text pages already)

Note: further use cases for processing local, deployment-specific trust anchors and credentials do also exist. They are identified and their solution is described in section 6.2.

4.2.2 Design

4.2.2.1 Overview

The solution for the imprinting cases 4.2.1 uses messages, data models and data stores according to RFC 6241 (NETCONF), RFC 7950 (YANG) and RFC 8342 (NMDA).

The following adaptation of figure in section 1.1 of RFC 6241 provides a conceptual partitioning that is used to describe the design of the imprinting solution:

```
              Layer                           Artifacts
          +--------------------+      +---------------------------------------+
      (4) |      Content       |      |       Credentials, trust anchors      |
          +--------------------+      +---------------------------------------+
                     |                                   |
          +--------------------+      +---------------------------------------+
      (3) |     Operations     |      |        <edit-config>, <commit>        |
          +--------------------+      +---------------------------------------+
                     |                                   |
          +--------------------+      +---------------------------------------+
      (2) |      Messages      |      |          <rpc>, <rpc-reply>           |
          +--------------------+      +---------------------------------------+
                     |                                   |
          +--------------------+      +---------------------------------------+
      (1) |  Secure Transport  |      |                  TLS                  |
          +--------------------+      +---------------------------------------+
```

4.2.2.2 Secure Transport

RFC 7589 describes the secure transport for NETCONF/YANG exchanges using TLS. The imprinting cases 4.2.1 require specific processing steps that are not covered by RFC 7589. Generalizations of RFC 7589 for the imprinting cases 4.2.1 are described in section 4.1.

4.2.2.3 Messages

RFC 6241 defines the messages in NETCONF/YANG exchanges for the imprinting cases 4.2.1.

4.2.2.4 Operations

Following NETCONF operations are used for the imprinting cases 4.2.1:

- `imprintTrustAnchor`: <edit-config> and <commit> (see 4.2.3 for details)
- `imprintCredential`: <edit-config> and <commit> (see 4.2.3 for details)

4.2.2.5 Content

Following YANG modules are used for the imprinting cases 4.2.1 as well as to access LDevID and IDevID credentials and trust anchors:

- `ietf-truststore` (see [16]): YANG module for trust anchor objects
- `ietf-keystore` (see [17]): YANG module for credential objects

RFC 8342 defines the handling of configuration (`<startup>`, `<candidate>`, `<running>`, `<intended>`) as well as operation state data stores (`<operational>`). This framework also applies to objects in `ietf-truststore` and `ietf-keystore` modules as illustrated by following adaptation of figure 2 in RFC 8342:

```
           +-------------+                  +-----------+
LDevID ➔   | <candidate> |<---+        +--->| <startup> |
           +-------------+    |        |    +-----------+
                   |          |        |           |
                   |        +-----------+          |
                   +-------->| <running> |<---------+
                            +-----------+
                                   |
                                   v
                            +------------+
                            | <intended> |
                            +------------+
                                   |
             dynamic               |   +-- learned configuration
             configuration         |   +-- system configuration
             datastores -----+     |   +-- default configuration
                             |     |   |
                             v     v   v
                           +---------------+
                           | <operational> | <-- system state ← IDevID
                           +---------------+
```

4.2.2.5.1 Trust Anchors

Trust anchors are accessed by the **`truststore`** container of the `ietf-truststore` module ([16] and https://www.yangcatalog.org/yang-search/yang_tree/ietf-truststore@2021-05-18):

- This container can hold 0..n CA trust anchors (from LDevID and IDevID domains)
- Individual CA certificate objects in the **`truststore`** are
  - Identified by their name. Well-known names (an enumeration defined by IEC/IEEE 60802) shall be used to distinguish individual items.
  - Represented as a data object of type “trust-anchor-cert-cms” (see [18])
- To authenticate other system entities e.g. TDMEs, an IA component uses the **`truststore`** incarnation **`operational.`**
- For LDevID trust anchor imprinting the **`truststore`** incarnation **`candidate`** is used[16].
- RFC 8342 specifies the transition from **`candidate`** to **`operational`**.

4.2.2.5.2 Credentials

Credentials are accessed by the **`keystore`** container of the `ietf-keytstore` module ([17] and https://www.yangcatalog.org/yang-search/yang_tree/ietf-keystore@2021-05-18):

- This container can hold 0..n credential objects (from LDevID and IDevID domains)
- Individual credential objects in **`keystore`** are
  - Identified by their name. Well-known names (an enumeration defined by IEC/IEEE 60802) shall be used to distinguish individual items.

16 IDevID trust anchor imprinting is out-of-scope for IEC/IEEE 60802

- Their certificate portion is represented as a data object of type “end-entity-cert-cms” (see [18])

- To authenticate itself against other system entities e.g., TDMEs, an IA component uses the **keystore** incarnation **operational**.

- For LDevID credential imprinting phase the **keystore** incarnation **candidate** is used[17].

- RFC 8342 specifies the transition from **candidate** to **operational**.

4.2.2.5.3 Prototype Messages

4.2.2.5.3.1 Imprint Trust Anchor

4.2.2.5.3.1.1 Plain Form

An example message for writing a trust anchor to the `candidate` configuration (see [16]):

```
<rpc message-id="001" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
         <target>
           <candidate/>
         </target>
         <config>
           <truststore xmlns=”urn:ietf:params:xml:ns:yang:ietf-truststore”>
             <certificate-bags>
               <certificate-bag>
                 <name>LDevID Bag</name>
                 <certificate>
                    <name>LDevID-NETCONF</name>
                    <cert-data>X509CaCertificateInPlainEnvelope</cert-data>
                 </certificate>
               </certificate-bag>
             </certificate-bags>
           </truststore>
         </config>
    </edit-config>
</rpc>
```

This prototype uses following specific items:

- `message-id` attribute: specific value but nothing special (could be any other value in the allowed value range)
- `name` values: specific value with a special purpose (well-known value from an IEC/IEEE 60802-specified enumeration to identify the scope of the given object).
- `cert-data` value: specific value of type “trust-anchor-cert-cms” providing a CA certificate enveloped in Base64-encoded CMS SignedData in degenerated form “certs-only” (no signature value) but nothing special (could be any other value in the allowed range)

TODO: generalize from single to multiple trust anchors for different purposes and domains. Also consider the naming concept in context of these multiple purposes and domains

4.2.2.5.3.1.2 Protected Form

A proposal for an example message for writing a protected trust anchor to the `candidate` configuration (not yet covered by [16]):

```
<rpc message-id="001” xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <asymmetric-keys xmlns="http://example.com/ns/example-crypto-types-
usage">
      <asymmetric-key>
        <name>LDevID-NETCONF</name>
```

[17] IDevID credential imprinting is out-of-scope for IEC/IEEE 60802

```
        <consume-voucher xmlns="urn:iec_ieee:tsn-ia:security">
          <voucher-data>rfc8366Voucher</voucher-data>
        </consume-voucher>
      </asymmetric-key>
    </asymmetric-keys>
  </action>
</rpc>
```

This prototype uses following specific items:

- message-id attribute: as above
- name value: as above
- xmlns value: **urn:iec_ieee:tsn-ia:security** refers to an own namespace for TSN-IA security for following elements:
    - **consume-voucher**: specific action to trigger the IA component to validate an RFC 8366 voucher object and store it the candidate configuration (if okay)
    - **voucher-data**: specific element providing a CA certificate in protected form.

  Important: using an own namespace is just an interim (➔ contribute to IETF)

Note: this proposal utilizes voucher object as specified by RFC 8366. An alternative form factor for the protected imprinting of trust anchors could be CMS SignedData (non-degenerated form) as specified in RFC 5652 (not shown above).

Open issues:

- Should 60802 support the imprinting of trust anchors in protected form (in addition to plain form aka TOFU)
- If yes: should this be based on RFC 8366 objects (aka vouchers) and/or CMS SignedData (non-degenerated form)
- If yes: revisit resp. align the above rough-upfront syntax proposal to carry trust anchors in protected form. Instead of an action this could also take the form of a feature e.g. ‘protected-trust-anchor’ (or ‘protected-certificate’ in addition to ‘certificate’)
- When done: make a proposal towards IETF to obviate a need for 60802-specific elements

4.2.2.5.3.2 Imprint Credential

4.2.2.5.3.2.1 External Key Generation

An example message for writing a credential with externally generated key pair to the candidate configuration (see [17]):

```
<rpc message-id="001" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
         <target>
           <candidate/>
         </target>
         <config>
           <keystore xmlns=”urn:ietf:params:xml:ns:yang:ietf-keystore”
                     xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-
                     types">
             <asymmetric-keys>
               <asymmetric-key>
                 <name>LDevID-NETCONF</name>
                 <public-key-format>ct:subject-public-key-info-format
                 </public-key-format>
                 <public-key>base64EncodedPubKey</public-key>
                 <private-key-format>TODO</private-key-format>
                 <cleartext-private-key>base64EncodedPrivKey
                 </cleartext-private-key>
                 <certificates>
                    <certificate>
                      <name>EE Certificate</name>
                      <cert-data>X509EeCertificateAndPathInEnvelope</cert-
data>
```

```
                    </certificate>
                 </certificates>
               </asymmetric-key>
             </asymmetric-keys>
           </keystore>
         </config>
    </edit-config>
</rpc>
```

TODO: generalize from single to multiple credentials for different purposes and domains. Also consider the naming concept in context of these multiple purposes and domains

This prototype uses following specific items:

- `message-id` attribute: as above
- `name` values: as above
- `private-key-format` value: dedicated value with a specific purpose; refers to the type and structure of a private key. Details depend on [18] and the cryptographic algorithm catalogue for TSN-IA (TBD).
- `cleartext-private-key` value: the private key in plain form[18]
- `public-key` value: the corresponding public key (also contained as SubjectPublicKeyInfo in the corresponding EE certificate)
- `cert-data` values: specific value of type “end-entity-cert-cms” providing an EE certificate and its intermediate CA certificate chain enveloped in Base64-encoded CMS SignedData in degenerated form (no signature value) but nothing special (could be any other value in the allowed range)

4.2.2.5.3.2.2 Internal Key Generation

Example messages for writing a credential with internally generated key pair to the `candidate` configuration. This subcase uses two exchanges.

Exchange 1: trigger the action "generate-certificate-signing-request" (see [18])

Request:

```
<rpc message-id="001” xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <asymmetric-keys xmlns="http://example.com/ns/example-crypto-types-
usage">
      <asymmetric-key>
        <name>LDevID-NETCONF</name>
        <generate-certificate-signing-request>
          <csr-info>base64EncodedPkcs10CertificationRequestInfo</csr-info>
        </generate-certificate-signing-request>
      </asymmetric-key>
    </asymmetric-keys>
  </action>
</rpc>
```

This request prototype uses following specific items:

- `message-id` attribute: as above
- `name` value: as above
- `csr-info` value: specific value of type Base64-encoded PKCS#10 `CertificationRequestInfo` (RFC 2986)[19] but nothing special (be any other value in the allowed range)

---

[18] The alternative is: <encrypted-private-key>. The option <cleartext-private-key> was picked to make a first description as simple as possible. This is not meant as the recommended or preferred form. Subsequent versions will elaborate on supported forms and their recommendation level for TSN-IA.

[19] Note: the `CertificationRequestInfo` child element SubjectPublicKeyInfo contains algorithm information and actual public key. The public key is empty when triggering the action "generate-certificate-signing-request"

Caveat: what is the correct interpretation of section-3.2 of [18] ("No Support for Key Generation")? A clarification is needed

The IA component internal processing steps that are triggered by this action are:

1) Receive and process the NETCONF request message (see above)
2) Base64-decode the <csr-info> value and parse it as a PKCS#10 `CertificationRequestInfo` object
3) Randomly generate a key pair for the specified algorithm (this information is provided as part of `SubjectPublicKeyInfo` in the PKCS#10 `CertificationRequestInfo`)
4) Internally store the private key together with its metadata e.g., algorithm information, <name> value in a secure manner
5) Put the public key into the (parsed) PKCS#10 `CertificationRequestInfo`
6) Serialize the PKCS#10 `CertificationRequestInfo` (including the public key)
7) Use the private key to create signature value for the (serialized) PKCS#10 `CertificationRequestInfo` (including the public key)
8) Construct a PKCS#10 `CertificationRequest` and Base64-encode it
9) Construct and send the NETCONF response message (see below)

Response:

```
<rpc-reply message-id="001"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <certificate-signing-request
     xmlns="http://example.com/ns/example-crypto-types-usage">
     base64EncodedPkcs10CertificationRequest
   </certificate-signing-request>
</rpc-reply>
```

This request prototype uses following specific items:

- `message-id` attribute: as above
- `certificate-signing-request` value: specific value of type Base64-encoded PKCS#10 `CertificationRequest` (RFC 2986) but nothing special (be any other value in the allowed range)

TODO: consider using NETCONF notifications to decouple the CSR supply in a response from its request (key pair generation may take some time)

Exchange 2: supply EE certificate and (opt.) intermediate sub-CA certificates (see [17])

```
<rpc message-id="002" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
         <target>
           <candidate/>
         </target>
         <config>
           <keystore xmlns=”urn:ietf:params:xml:ns:yang:ietf-keystore”
                     xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-
                     types">
             <asymmetric-keys>
               <asymmetric-key>
                 <name>LDevID-NETCONF</name>
                 <public-key-format>ct:subject-public-key-info-format
                 </public-key-format>
                 <public-key>base64EncodedPubKey</public-key>
                 <private-key-format>TODO</private-key-format>
                 <hidden-private-key/>
                 <certificates>
                    <certificate>
                      <name>EE Certificate</name>
```

```
                      <cert-data>X509EeCertificateAndPathInEnvelope</cert-
data>
                    </certificate>
                 </certificates>
               </asymmetric-key>
             </asymmetric-keys>
           </keystore>
         </config>
    </edit-config>
</rpc>
```

This prototype uses following specific items:

- `message-id` attribute: as above
- `name` values: as above
- `public-key` value: as above
- `cert-data` values: as above

4.2.3 Illustration

This chapter illustrates the imprinting and use of LDevID-NETCONF credentials and trust anchors. This description is informational and focusses on following “sunshine”:

- Step 1: Booting with IDevID
- Step 2: Imprinting of Trust Anchor for LDevID-NETCONF
- Step 3: Imprinting of LDevID-NETCONF Credential
- Step 4: Operationalizing LDevID-NETCONF
- Step 5: Using LDevID-NETCONF

4.2.3.1 Step 1: Booting with IDevID

When an IA component boots with its factory defaults, following **`truststore`** and **`keystore`** incarnations become available (see RFC 8342 as well as sections 3 [16] and [17]):

- `truststore`
    - Configuration stores:
        - `startup`: ---
        - `candidate`: ---
        - `running`: ---
        - `intended`: ---
    - `operational`: trust anchor for IDevIDs (not persisted across reboots)
- `keystore`
    - Configuration stores:
        - `startup`: ---
        - `candidate`: ---
        - `running`: ---
        - `intended`: ---
    - `operational`: IDevID credential (not persisted across reboots)

TODO: propose a naming convention to allow the IDevID credential and trust anchor to be found inside the truststore and keystore

4.2.3.2 Step 2: Imprinting of Trust Anchor for LDevID-NETCONF

When an IA component gets imprinted with the trust anchor for LDevID-NETCONF, the only trust anchor that is available in `operational` allows to validate IDevID credentials. The imprinting client cannot be assumed to be equipped with IDevIDs. This gap is addressed by a specific procedure called “provisional accept of client cert” described above. Following **`truststore`** and **`keystore`** incarnations become available through this imprinting step (see 4.2.2.5.3.1 for the request that triggers this state change).

- `truststore`
  - Configuration stores:
    - `startup`: ---
    - `candidate`: trust anchor for LDevIDs (not persisted across reboots)
    - `running`: ---
    - `intended`: ---
  - `operational`: trust anchor for IDevIDs (not persisted across reboots)
- `keystore`
  - Configuration stores:
    - `startup`: ---
    - `candidate`: ---
    - `running`: ---
    - `intended`: ---
  - `operational`: IDevID credential (not persisted across reboots)

Note: this imprinting step uses step 1 stores as follows:

- Trust anchor for IDevIDs in the **`truststore`** incarnation **`operational`**: not used for unprotected imprinting (TOFU), used for validating the to-be-imprinted payload object (voucher) for protected imprinting. In any case: not used for TLS client authentication.
- IDevID credential in the **`keystore`** incarnation **`operational`**: used for TLS server authentication

4.2.3.3 Step 3: Imprinting of LDevID-NETCONF Credential

When an IA component gets imprinted with its LDevID-NETCONF credential directly after step 2, the only trust anchor that is available in `operational` allows to validate IDevID credentials. This gap can be addressed by continuing to use the TLS session established for step 2 during step 3 (if this can or shall not happen then the trust anchor for LDevID shall be propagated to **`operational`** before imprinting the LDevID-NETCONF credential). Following **`truststore`** and **`keystore`** incarnations become available through this procedure (see 4.2.2.5.3.2 for the request that triggers this state change):

- `truststore`
  - Configuration stores:

            - `startup`: ---
            - `candidate`: trust anchor for LDevIDs (not persisted across reboots)
            - `running`: ---
            - `intended`: ---
    - `operational`: trust anchor for IDevIDs (not persisted across reboots)
- `keystore`
    - Configuration stores:
        - `startup`: ---
        - `candidate`: LDevID-NETCONF credential (not persisted across reboots)
        - `running`: ---
        - `intended`: ---
    - `operational`: IDevID credential (not persisted across reboots)

Note: this imprinting step does not rely on step 2 additions (not yet operational) on application-level but relies on step 2 processing ("provisional accept of client cert") on TLS-level.

4.2.3.4 Step 4: Operationalizing LDevID-NETCONF

By standard means (NETCONF <commit> operation) according to RFCs 6241/7950/8342, the LDevID-NETCONF credential and trust anchor are operationalized. Following **`truststore`** and **`keystore`** incarnations become available through this procedure:

- `truststore`
    - Configuration stores:
        - `startup`: ---
        - `candidate`: ---
        - `running`: trust anchor for LDevIDs (persisted across reboots)
        - `intended`: trust anchor for LDevIDs (persisted across reboots)
    - `operational`: trust anchor for LDevIDs and IDevIDs (not persisted across reboots)
- `keystore`
    - Configuration stores:
        - `startup`: ---
        - `candidate`: ---
        - `running`: LDevID-NETCONF credential (persisted across reboots)
        - `intended`: LDevID-NETCONF credential (persisted across reboots)

    - `operational`: LDevID-NETCONF and IDevID credentials (not persisted across reboots)

4.2.3.5 Step 5: Using LDevID-NETCONF

After step 4 LDevID-NETCONF credential and trust anchor can be used by the IA component for NETCONF-over-TLS according to RFC 7589. This happens as follows:

- Trust anchor for LDevID-NETCONF:
    - Is obtained from the **`truststore`** incarnation **`operational`**
    - Is found by its well-known name **`LDevID-NETCONF`** inside the **`LDevID Bag`**
    - Is used for sending out the TLS **`CertificateRequest`** message
    - Is used for processing the TLS **`Certificate`** message sent by the client
- LDevID-NETCONF credential:
    - Is obtained from the **`keystore`** incarnation **`operational`**
    - Is found by its well-known name **`LDevID-NETCONF`**
    - Is used for sending out the TLS **`Certificate`** message
    - Is used for processing specific TLS messages (details depend on the employed cipher suite which is again a subject to the cryptographic algorithm catalogue for IEC/IEEE 60802 [TODO]) sent by the NETCONF client

4.2.3.6 Other Processing Steps

TODO: discuss further processing steps e.g., reboot and reset-to-factory (note: this relates to the TSN-IA use cases)

5 Using the Solution – With Respect To NETCONF/YANG

5.1 Resource Access Authorization for NETCONF/YANG

5.1.1 Access Control Mechanism

On the mechanism level, IEC/IEEE 60802 uses NACM (RFC 8341) for the access control to NETCONF/YANG resources. NACM especially specifies a YANG data model (`ietf-netconf-acm`) for expressing rules to control access to NETCONF/YANG resources. The corresponding container is called **`nacm`**.

The NACM design pattern strikes with following properties:

i. NACM access enforcement uses configurable rules that live on the same server which is protected by NACM access enforcement.

ii. NACM rules are managed through the same instance of the channel that NACM protects.

This deviates from typical access control approaches in IT. It requires NACM to be self-reflexive: *capable of expressing and enforcing rules about changing itself*. This property is a key enabler for IEC/IEEE 60802 security especially its resource access authorizations.

5.1.2 Access Control Model

On the conceptual level, IEC/IEEE 60802 profiles NACM to deliver a role-based authorization model (RBAC)[20]. This role-based model is characterized by:

- The set of NETCONF/YANG resources upon a system component is partitioned according to its YANG modules. Each item in this partitioning e.g., `ietf-truststore` is assigned one or more permission e.g., "Write". The resulting ensemble is assigned a permission name e.g., "PermitWrite".
    - IEC/IEEE 60802 security shall specify the set of permission names for IEC/IEEE 60802 (TODO)
- The set of system actors is assigned one or more roles e.g., "TruststoreAdminRole".
    - IEC/IEEE 60802 shall specify the set of role names as well as the mechanism to determine the role names that are assigned to an actor (see 5.1.4). An initial drop for this is (TODO: consider further roles [there should not be too many]):
        - `TruststoreAdminRole`
        - `KeystoreAdminRole`
        - `UserMappingAdminRole`
        - `NACMAdminRole`
        - `RecoverySessionRole`
        - `CommitRole`
        - `ResetToFactoryRole`
    - IEC/IEEE 60802 does not specify the assignment of role names to actual system entities. This is a duty of system owners or operators.
- The role names get assigned to permissions, so that a system actor is authorized to perform an action upon a resource provided a role name is assigned to it that encompasses this action upon this resource e.g., the permission "PermitWrite" for the `truststore` container is assigned to the "TruststoreAdminRole".
    - IEC/IEEE 60802 shall specify the role to permission assignment. An initial drop for this is (TODO: consider further roles):
        - `TruststoreAdminRole:` clients with this role can write to `truststore` container (subject to details,LDevID vs. IDevID)
        - `KeystoreAdminRole:` clients with this role can write to the `keystore` container (subject to details, LDevID vs. IDevID)
        - `UserMappingAdmin:` clients with this role can write to the `x509c2n` container
        - `NACMAdminRole:` clients with this role can write to the `nacm` container (subject to details, IEC/IEEE 60802 vs. custom rules)
        - `RecoverySessionRole:` clients with this role act according the NACM recovery session

———————————

[20] NACM does natively not deliver a role-based access control model but can be geared towards a role-based model by profiling

- `CommitRole:` clients with this role can perform the commit RPC
- `ResetToFactoryRole:` clients with this role can perform the factory-reset RPC (RFC 8808)

Background: RBAC is one of the well-known strategies to manage the complexity of the so-called access control matrix (x-axis: all system resources, y-axis: all system actors, x/y fields: the access rights). There are other conceptual approaches for modelling this conceptual matrix especially DAC and MAC. The role-based approach matches the needs of TSN-IA better than especially DAC or MAC.

5.1.3 NACM Access Control Rules

5.1.3.1 CRUDX for the `truststore` Container

NACM snippet that allows any authenticated client to read `ietf-truststore` contents and authenticated clients with ‘TruststoreAdminRole’ to write `ietf-truststore` contents:

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <enable-nacm>true</enable-nacm>
  <read-default>deny</read-default>
  <write-default>deny</write-default>
  <exec-default>deny</exec-default>
  <enable-external-groups>false</enable-external-groups>
  <groups>
    <group>
      <name>TruststoreAdmin</name>
      <user-name>TruststoreAdminRole</user-name>
    </group>
    <!-- other group entries -->
  </groups>
  <rule-list>
    <name>PermitRead for all</name>
    <group>*</group>
    <rule>
      <name>PermitRead</name>
      <module-name>ietf-truststore</module-name>
      <access-operations>read</access-operations>
      <action>permit</action>
    </rule>
  </rule-list>
  <rule-list>
    <name>PermitWrite for TruststoreAdmin</name>
    <group>TruststoreAdmin</group>
    <rule>
      <name>PermitWrite</name>
      <module-name>ietf-truststore</module-name>
      <access-operations>create update delete</access-operations>
      <action>permit</action>
    </rule>
  </rule-list>
  <!-- other rule-list entries -->
</nacm>
```

TODO: refinements (LDevID vs IDevID)

5.1.3.2 CRUDX for the Certificate-to-Name Mapping Container

NACM snippet that allows any authenticated client to write `ietf-x509-cert-to-name` contents:

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <enable-nacm>true</enable-nacm>
  <read-default>deny</read-default>
```

```
  <write-default>deny</write-default>
  <exec-default>deny</exec-default>
  <enable-external-groups>false</enable-external-groups>
  <groups>
    <group>
      <name>UserMappingAdmin</name>
      <user-name>UserMappingAdminRole</user-name>
    </group>
    <!-- other group entries -->
  </groups>
  <rule-list>
    <name>PermitWrite for UserMappingAdmin</name>
    <group>UserMappingAdmin</group>
    <rule>
      <name>PermitWrite</name>
      <module-name>ietf-x509-cert-to-name</module-name>
      <access-operations>create update delete</access-operations>
      <action>permit</action>
    </rule>
  </rule-list>
  <!-- other rule-list entries -->
</nacm>
```

5.1.3.3 CRUDX for the `keystore` container

TODO: elaboration (LDevID vs IDevID, public vs. private portions)

5.1.3.4 CRUDX for the `nacm` Container

In order to be able to update the initial or current instance of the `nacm` container there shall be a NACM rule that allows one or more actors to manage the NACM rules.

NACM snippet that allows authenticated clients with '**NACMAdminRole**' to write `ietf-netconf-acm` contents:

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <enable-nacm>true</enable-nacm>
  <read-default>deny</read-default>
  <write-default>deny</write-default>
  <exec-default>deny</exec-default>
  <enable-external-groups>false</enable-external-groups>
  <groups>
    <group>
      <name>NACMAdmin</name>
      <user-name>NACMAdminRole</user-name>
    </group>
    <!-- other group entries -->
  </groups>
  <rule-list>
    <name>PermitAll for NACMAdminRole</name>
    <group>NACMAdmin</group>
    <rule>
      <name>PermitAll</name>
      <module-name>ietf-netconf-acm</module-name>
      <access-operations>*</access-operations>
      <action>permit</action>
    </rule>
  </rule-list>
  <!-- other rule-list entries -->
</nacm>
```

TODO: refinements (IEC/IEEE 60802 rules (read-only) vs. manufacturer or owner/operator extensions (read-write))

5.1.3.5 CRUDX for <commit>

NACM snippet that allows authenticated clients with ‘**CommitRole**’ to execute <commit>:

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <enable-nacm>true</enable-nacm>
  <read-default>deny</read-default>
  <write-default>deny</write-default>
  <exec-default>deny</exec-default>
  <enable-external-groups>false</enable-external-groups>
  <groups>
    <group>
      <name>Committers</name>
      <user-name>CommitRole</user-name>
    </group>
    <!-- other group entries -->
  </groups>
  <rule-list>
    <name>Permit for CommitRole</name>
    <group>Committers</group>
    <rule>
      <name>PermitCommit</name>
      <rule-type>
        <protocol-operation>
          <rpc-name>commit</rpc-name>
        </protocol-operation>
      </rule-type>
      <action>permit</action>
    </rule>
  </rule-list>
  <!-- other rule-list entries -->
</nacm>
```

5.1.3.6 CRUDX for <factory-reset>

NACM snippet that allows authenticated clients with ‘**FactoryResetRole**’ to execute <factory-reset>:

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <enable-nacm>true</enable-nacm>
  <read-default>deny</read-default>
  <write-default>deny</write-default>
  <exec-default>deny</exec-default>
  <enable-external-groups>false</enable-external-groups>
  <groups>
    <group>
      <name>FactoryResetters</name>
      <user-name>FactoryResetRole</user-name>
    </group>
    <!-- other group entries -->
  </groups>
  <rule-list>
    <name>Permit for FactoryResetRole</name>
    <group>Committers</group>
    <rule>
      <name>FactoryResetters</name>
      <rule-type>
        <protocol-operation>
          <rpc-name>factory-reset</rpc-name>
        </protocol-operation>
      </rule-type>
      <action>permit</action>
    </rule>
  </rule-list>
```

```
  <!-- other rule-list entries -->
</nacm>
```

5.1.3.7 CRUDX for Other NETCONF/YANG Resources

TODO: is there a catalogue of YANG modules that are required for or supported by IEC/IEEE 60802? Are there qualifications for the items in this catalogue e.g. mandatory/optional or read-only for owner/operator vs. write by owner/operator?

5.1.4 NETCONF Usernames

RFC 7589 (section 7) requires NETCONF servers to map client certificates to “NETCONF usernames” and specifies a concrete mapping procedure for this purpose. Note:

- This is defined as part of the binding between NETCONF and TLS (RFC 7589).
- It happens outside the scope of the applicable TLS specification (RFC 5246).

This mapping is represented by the YANG module `ietf-x509-cert-to-name`.

The cert-to-name mapping procedure in RFC 7589 (section 7) is used as follows by IEC/IEEE 60802:

(a) is profiled to comprise mapping list with a single entry containing:
  - `fingerprint`: the fingerprint of the trust anchor for the production cell or site
  - `map_type`: `common-name`[21]

(b) will produce a match in its subclause 2 provided the following holds
  - Trust anchor is not self-signed: always matches for clients that
    - i. Possess a valid EE certificate (and chain) issued underneath the root CA certificate that is identified by the `fingerprint` value in (a)
    - ii. Can demonstrate PoP for the private key that matches the public key in its EE certificate
  - Root CA certificate is self-signed: matches when i and ii hold and when the root CA certificate is part of the TLS `Certificate` message (this is allowed by the TLS specification but deviates from the TLS best practices, see 5.2.1)

(c) will provide the CN portion in the subject DN as the NETCONF username. IEC/IEEE 60802 profiles this string value to carry one or more of the IEC/IEEE 60802-defined role names e.g., “**`TruststoreAdminRole`**” (multiple role names in one CN value are separated by whitespace), not an actual username e.g., “John Doe”.

(d) as-is (never applies in sunshine case)

The small print for this profile of the client identity mapping procedure in RFC 7589 is:

- Confined to the X.500 naming concept, which is actually deprecated by RFC 7589 “*The usage of CommonNames is deprecated and users are encouraged to use subjectAltName mapping methods instead.*”
- Requires elaborating on DN name building rules beyond their sub-portion; different system actors must have different DN values but can have the same CN value

Resolution options for this issue of type “*would work but is somewhat phoney*”:

[21] Alternatives: ‘specified’ would require multiple items (one item per role). ‘san-rfc822-name’, ‘san-dns-name’, ‘san-ip-address’ and ‘san-any’ have issues with syntax/semantics in case of a role-based access control model

- Getting rid of X.500 naming: an additional mapping e.g., ‘san-60802-role’ (60802 role in subject alternative name) or ‘ext-60802-role’ (60802 role in own, private extension; this is preferred over ‘san-60802-role’ [using ASN.1 “GeneralName” for carrying role assignments is syntactically possible {OtherName} but would be semantically misleading])
- Getting rid of the TLS best practices violation: modified mapping procedure

TODO: are we allowed to propose another certificate-to-NETCONF username mapping type or even another mapping strategy?

Important:

- The IEC/IEEE 60802 roles that are assigned to a system actor (and that are determined by the client certificate to NETCONF username mapping) are used for determining its resource access authorization.
- The IEC/IEEE 60802 roles are not used for auditing/logging purposes. Audit/logging uses: subject DN in the EE certificate (X.500 naming concept) and/or SAN in the EE certificate (IETF naming concept that is not confined to the X.500 straitjacket)

  Note: EE certificates that are used by IEC/IEEE 60802 are not related to human users. Hence PII resp. privacy is a non-issue in IEC/IEEE 60802.

TODO: the client identity mapping in RFC 7589 appears to be overhauled with current IETF drafts, see https://datatracker.ietf.org/doc/html/draft-ietf-netconf-netconf-client-server-23#section-3.3, keyword “cert-to-name-mapping”

5.1.5 Processing Pipeline

The processing pipeline for NETCONF/YANG exchanges in IEC/IEEE 60802 has 4 main steps. These steps are done by the system component that shall be configured i.e., acts in NETCONF server role:

1. Establish TLS session with mutual entity authentication using option a or b:
   a. *Taking-off case*[22] (see chapter 4 for details):
      - IDevID credential and trust anchor on server side
      - LDevID-NETCONF credential and trust anchor on client side
   b. *Cruising case*[23] (see 5.2.1 for details):
      - LDevID-NETCONF credential and trust anchor on server side
      - LDevID-NETCONF credential and trust anchor on client side
2. (If step 1 was successful): determine the NETCONF username of the client (see 5.1.4)
3. (If step 2 was successful): enforce the permissions of the client (see 5.1.3)
4. (If step 3 was permitted): perform the requested NETCONF/YANG operation

These steps depend on specific items in the operational and configuration data stores:

- Step 1: uses contents of the YANG modules `ietf-truststore` and `ietf-keystore`

---

[22] Before the imprinting of LDevID-NETCONF credentials and trust anchor to the system component that acts in NETCONF and TLS server role has happened

[23] After the imprinting of LDevID-NETCONF credentials and trust anchor to the system component that acts in NETCONF and TLS server role has happened

- Step 2: uses contents of the YANG module `ietf-x509-cert-to-name` (IEC/IEEE 60802-specific certificate to NETCONF username mapping)
- Step 3: uses contents of the YANG module `ietf-netconf-acm` (IEC/IEEE 60802-specific NACM rules, opt. custom additions)
- Step 4: can update the contents of the YANG modules used in steps 1, 2 and 3

This presents a repercussion: step 4 can change the operational context for steps 1/2/3 that step 4 depends upon. This requires an explicit and granular consideration of booting with factory defaults and resetting to factory.

5.1.5.1 Booting with Factory Default

- Operational (and configuration) data stores state after booting with factory defaults:
    - `ietf-truststore`: manufacturer created and built-in trust anchor for verifying IDevID credentials (read-only)
    - `ietf-keystore`: manufacturer created and built-in IDevID credential (read-only)
    - `ietf-x509-cert-to-name`: empty[24]
    - `ietf-netconf-acm`: IEC/IEEE 60802-defined and manufacturer built-in NACM rules (read-only)
    - Other YANG modules: any initial state as specified by IEC/IEEE 60802
- Imprinting sequence - *Taking-off* phase:
    1. Imprint (<edit-config>) the trust anchor for verifying LDevID(-NETCONF) credentials using the NACM “recovery session” feature:
        - Step 1, subcase a: challenge the client for authentication according to any trust anchor of its choice (empty `certificate_authorities` portion in TLS `CertificateRequest)` and establish this wildcard authentication. Details are explained in 4.1.1 (tasks for the TLS client) and 4.1.2 (tasks for the TLS server). Note: the latter is conducted according a “provisional accept of client cert” procedure: any (valid) credential with role “RecoverySessionRole” allows clients to pass this step 1. This is true for “official” as well as for “rogue” credentials (see protected vs. unprotected imprinting for consequences and their mitigations)
        - Step 2: no cert-to-role mapping happens for the “recovery session” feature of NACM. Any (valid) EE certificate with role “RecoverySessionRole” allows clients to pass step 2
        - Step 3: no NACM enforcement happens for the “recovery session” feature of NACM. Any ((valid) EE certificate with role “RecoverySessionRole” allows clients to pass step 3 (has caveats, see above)
        - Step 4: depends on whether the trust anchor imprinting is protected or not
            - Unprotected imprinting: no further security checks possible. The provided trust anchor is configured into the `truststore`. Does not change of the described exposition TODO: identify resulting attack vectors, discuss their severity

———————

[24] The fingerprint of the LDevID trust anchor is not known at this point in time

  - Protected imprinting: signature checks using the built-in trust anchor for IDevID. The provided trust anchor is only configured into the `truststore` when these checks are passed. Can reduce the described exposition: a (valid) voucher is needed to pass step 4 (if RFC 8366 is used for protected imprinting). TODO: identify resulting price tag, discuss its affordability

2. Imprint (<edit-config>) instruction(s) for the mapping from client certificates to NETCONF usernames using the “recovery session” feature of NACM:
   - Step 1, subcase a: as above[25]
   - Step 2: as above
   - Step 3: as above
   - Step 4: the provided mapping is configured into the `x509c2n`. Any (valid) credential for the role “RecoverySessionRole” allows clients to pass steps 1-4.
3. Operationalize (<commit>) the configuration changes 1 and 2 using the “recovery session” feature of NACM:
   - Step 1, subcase a: as above[26]
   - Step 2: as above
   - Step 3: as above
   - Step 4: operationalize the configuration changes. Any (valid) credential for the role “RecoverySessionRole” allows clients to pass steps 1-4

Note: subsequent `truststore` and `x509c2n` operations during the *cruising* phase can be performed without depending on the NACM ‘recovery session’ resp. the role IEC/IEEE 60802 “RecoverySessionRole”. To avoid an excessive use of the NACM ‘recovery session’ the IEC/IEEE 60802 “RecoverySessionRole” should not be used as part of multi-valued role assignments.

- Imprinting sequence - *Crusing* phase:

4. Imprint (<edit-config>) the LDevID-NETCONF credential using the KeystoreAdminRole:
   - Step 1, subcase b: challenge the client for authentication according to the trust anchor for LDevID(-NETCONF) credentials and establish this
   - Step 2: extracts the common-name value in the EE certificate. Any LDevID-NETCONF EE certificate with DN/CN value passes (the certificate fingerprint matching is covered by step 1)
   - Step 3: uses 5.1.3.3 to check the common-name value against “KeystoreAdminRole”. Any LDevID-NETCONF EE certificate with “KeystoreAdminRole” passes.
   - Step 4: depends on whether key pair is generated locally (2 subsequent NETCONF/YANG exchanges for LDevID credential imprinting) or remotely (1 exchange for LDevID credential imprinting). Any (valid) credential for the role “KeystoreAdminRole” issued by the trust anchor that was imprinted in

[25] Using the trust anchor for LDevID(-NETCONF) credentials requires to make it operational

[26] Using the trust anchor for LDevID(-NETCONF) credentials requires to make it operational

1, employed for NETCONF username mapping in 2 and operationalized in 3 allows clients to pass steps 1-4.

5. Supply custom NACM rules (optional) using the NACMAdminRole:
    - Step 1, subcase b: as above.
    - Step 2: as above
    - Step 3: uses 5.1.3.4 to check the common-name value against “NACMAdminRole”. Any LDevID-NETCONF EE certificate with “NACMAdminRole” passes. Also makes sure write operations do not affect the basic NACM rules specified by IEC/IEEE 60802
    - Step 4: straight-forward (when steps 1-3 are passed).
6. Operationalize (<commit>) the configuration changes 5 and 6 using the ConfigurationManagerRole
    - Step 1, subcase b: as above
    - Step 2: as above
    - Step 3: uses 5.1.3.5 to check the common-name value against “ConfigurationManagerRole”. Any LDevID-NETCONF EE certificate with “ConfigurationManagerRole” passes.
    - Step 4: operationalize the configuration changes. Any (valid) credential for the role “ConfigurationManagerRole” allows clients to pass steps 1-4

- Operational (and configuration) data state stores after imprinting:
    - `ietf-truststore`: owner/operator configured trust anchor for LDevID(-NETCONF) credentials. Manufacturer created and built-in trust anchor for verifying IDevID credentials (read-only)
    - `ietf-x509-cert-to-name`: owner/operator created configuration instance of the IEC/IEEE 60802-defined cert-to-name mapping
    - `ietf-keystore`: owner/operator configured LDevID(-NETCONF) credential. Manufacturer created and built-in IDevID credential (read-only)
    - `ietf-netconf-acm`: owner/operator configured NACM rules (optional). IEC/IEEE 60802-defined and manufacturer built-in NACM rules (read-only).
    - Other YANG modules: arbitrary

5.1.5.2 Resetting to Factory Default

Resetting to factory shall be supported according to the means that are defined by RFC 8808. RFC 8808 defines a NETCONF action "factory-reset" and a corresponding YANG module `ietf-factory-default` for the purpose of factory reset.

IEC/IEEE 60802 components protect the NETCONF action "factory-reset" with the same approach as other NETCONF operations:

- Perform factory-reset using the `FactoryResetRole`:
    - Step 1: challenges the client for authentication with respect to the LDevID(-NETCONF) trust anchor. Any valid LDevID-NETCONF credential issued by the imprinted trust anchor passes.

  - Step 2: extracts the common-name value in the EE certificate. Any LDevID-NETCONF EE certificate with DN/CN value passes (the certificate fingerprint matching is covered by step 1)
  - Step 3: uses 5.1.3.6 to check the common-name value against “FactoryResetRole”. Any LDevID-NETCONF EE certificate with “FactoryResetRole” passes.
  - Step 4: straight-forward (when steps 1-3 are passed).

Supporting additional means for factory reset e.g., physical presence (reset button) is at the discretion of IA component manufacturers. The protection of such additional means is out-of-scope for IEC/IEEE 60802 security.

5.2 Message Exchange Protection for NETCONF/YANG

5.2.1 TLS Profile

- TLS 1.2 with mutual authentication (mandated by RFC 7589). See Annex D for more information.
- Cipher suites:
  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
  - Note: RFC 7589 implicitly mandates TLS_RSA_WITH_AES_128_CBC_SHA by referring to RFC 5246. IEC/IEEE 60802 deselects this cipher suite for following reasons: excessive asymmetric key lengths needed, no AEAD scheme, no PFS
- Curves for ECC:
  - NIST curves (NIST FIPS 186 ‘Digital Signature Standard (DSS)’):
    - secp521
    - secp256
  - Bernstein/Goldilocks curves (RFC 7748)
    - curve448 (“Goldilocks” aka “Edwards” curve)
    - curve25519 (“Bernstein” curve)
  - TODO: discussion needed with 802.1 Security Task Group: support of ECC with >200 bits security strength esp. secp521?
- PKI integration:
  - Certification paths:
    - Server: arbitrary length (1..n), self-signed root CA certificate shall be present in TLS `Certificate` messages (needed for 5.1.4, deviates from default behavior[27]). Implementations must support TLS `Certificate` message with 1..3 certificates objects .i.e. a PKI path length of 3.

[27] RFC 5246: *Because certificate validation requires that root keys be distributed independently, the self-signed certificate that specifies the root certificate authority MAY be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case.*

        - Client: ditto
    - Certificate contents: X.509v3 public key certificates according to RFC 5280 fulfilling the following criteria
        - Server: the EE certificate shall carry the FQDN of the NETCONF server in its subjectAltName extension (mandated by RFC 7589) and digital signature in its keyUsage extension. TODO: validity period (relates to certificate supply/update strategy)
        - Client: the EE certificate shall carry digitalSignature in its keyUsage extension. TODO: validity period (relates to certificate supply/update strategy)
    - Certificate supply/update strategy: TODO: informative text considering e.g. human operated and/or automated supply and update
    - Certificate revocation strategy: TODO: informative (preferred) and/or normative text
- TLS extensions (see RFC 6066): many RFC 6066 extensions assume TLS clients to be constrained and TLS servers to be rather unconstrained. This does not exactly match the IEC/IEEE 60802 preconditions.
    - Server name indication: not used (addresses single server instances that serve multiple DNS names)
    - Maximum fragment length negotiation: TODO (allows to agree on a max TLS record layer payload length shorter than 2**14)
    - Client certificate URLs: not used (allows to replace content by identifier in case of TLS `Certificate` messages sent by the client)
    - Trusted CA indication: not used (allows to clients to tell servers about their trust anchors)
    - Truncated HMAC: not used (allows to ask for truncating the output of the hash function to 80 bits when forming MAC tags)
    - Certificate status request: TODO (allows TLS clients to ask for OCSP rather than CRLs for verifying server certificates)

6 Exploiting the Solution – Other Trust Anchors and Credentials

6.1 Supply

TODO: describe the supply (creating) of local, deployment-specific trust anchors and credentials for other exchanges than NETCONF/YANG by means of NETCONF/YANG (the supply for NETCONF/YANG exchanges by means of NETCONF/YANG is described in 4)

6.2 Handling

TODO: describe the handling (using/updating/deleting…) of local, deployment-specific trust anchors and credentials for any exchanges by means of NETCONF/YANG.

7 Using the Exploitation – Beyond NETCONF/YANG

7.1 TSN-IA Defined Exchanges Beyond NETCONF/YANG

7.1.1 Resources Access Authorization

TODO: describe how the imprinting solution for TSN-IA-defined exchanges other than NETCONF/YANG can be exploited to protect the access to resources (exposed by these exchanges)

7.1.2 Message Exchange Protection

TODO: describe how the imprinting solution for TSN-IA-defined exchanges other than NETCONF/YANG can be exploited to protect the actual message exchanges

7.2 Other Exchanges

Using this exploitation is regarded a matter of middleware and application components. This needs to be elaborated by these specifications. It is not detailed by TSN-IA.

Annex A IEEE 802.1AR ‘Secure Device Identity’

A.1 IDevID Objects

- Abbreviation for: **I**nitial **Dev**ice **ID**entifier
- Definition (somewhat rephrased for simplicity): a manufacturer-generated and installed object that is cryptographically bound to the component, and that comprises (see [10] for all applicable details):
  - An asymmetric **private key**
  - An **EE certificate** which binds the corresponding public key to information about the component and that is stated by its manufacturer. This certificate is assumed to be:
    - Valid eternally (notAfter=99991231235959Z)
    - Have an X.500 subject field (DN) carrying a unique product serial number[28].
    - Not self-signed
  - A **certificate chain** i.e., a list of intermediate CA certificates that links the EE certificate to the trust anchor (self-signed root CA certificate) of the manufacturer
- Quantity: IEEE 802.1AR-2018 allows one component to possess one or more IDevIDs (IEEE 802.1AR-2009 did limit this to one IDevID).
- Important:
  - IDevID issuance and supply is meant to happen once in the lifetime of the component (during its manufacturing and before its shipment). Typically, the IDevID object is never updated or erased.
  - Since IDevID objects are created at component manufacturing time they can only contain information known at manufacturing time (these items are called ‘product master data’ herein).
  - System integrators and owner/operators do not have to worry about IDevID object production - they consume IDevIDs only.
  - Invalidation of an IDevID credential does not (have to) prevent the usage of the component:
    - This only prevents the use of this IDevID object. This affects usages of this IDevID after the invalidation event, not (or not necessarily) earlier usages of this IDevID before its invalidation event.
    - This does not affect the usage of other IDevID credentials - if there are multiple IDevID credential objects for a specific component.

A.2 LDevID Objects

- Abbreviation for: **L**ocally significant **Dev**ice **ID**entifier

---

[28] The `serialNumber` value shall be unique within the domain of significance that is identified by the issuer name, not just within the context of precursor DN fields in the subject name

- Definition (somewhat rephrased for simplicity): a system integrator or owner/operator-generated and installed object that is cryptographically bound to the component, and that comprises (see [10] for all applicable details):
    - An asymmetric **private key**
    - An **EE certificate** which binds the corresponding public key to information about the component and that is stated by its system integrator or owner/operator. This certificate is assumed to be:
        - Not eternal, no [notBefore, notAfter] interval length is suggested
        - Not self-signed
    - A **certificate chain** i.e., a list of intermediate CA certificates that links the EE certificate to the trust anchor (self-signed root CA certificate) of the system integrator or owner/operator.
- Quantity: IEEE 802.1AR-2009 and 2018 allow one component to possess one or more LDevIDs
- Important:
    - LDevID issuance and supply is meant to happen one or more times during the lifetime of the component (during bootstrapping or even operation phases). The LDevID objects can be updated or erased. A security model is needed to prevent attackers from supplying or managing LDevID objects.
    - The LDevID objects are created at bootstrapping or even operation time of the component. Hence, they can and shall contain information known when this component is bootstrapped or operated but which is not known when the component is manufactured (this is also called ‘deployment master data’ herein).
    - Manufacturers do not have to worry about LDevID supply. With respect to LDevIDs their “only” concern is supplying (protected and initially empty) storage and means to support system integrators and owners/operators e.g., building blocks for cryptographic operations such as random number generation, key pair generation, object signing and validating.
    - Invalidation of an LDevID credential does not (have to) prevent the usage of the component:
        - This only prevents the use of this LDevID credential. This affects usages of this LDevID credential after the invalidation event, not (or not necessarily) earlier usages of this IDevID before its invalidation event.
        - This does not affect the usage of other LDevID credentials - if there are multiple LDevID credential objects for a specific component.
        - Although this reads equivalent to the corresponding section for IDevIDs, the consequences of a LDevID invalidation are more severe than IDevID invalidation. This is due to following:
            - LDevIDs should be assumed to be used often (hint: “daily use”)
            - IDevIDs can be assumed to be used occasionally (hint: “annual use”)

Annex B IETF RFC 6125

RFC 6125 (see [6]) is mandated for checking the identity of a NETCONF-over-TLS server by RFC 7589 '*Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication*' (see [8]).

RFC 6125 requires the name of an application service to be (or to be based on) a DNS domain name in one of the following forms:

- **Traditional domain name**: a FQDN with labels constrained to ASCII letter, digits and hyphen (further small-print applies)
- **Internationalized domain name**: a FQDN with at least one Unicode label (further small-print applies)

Following 'actual vs. expected'-matching rules apply for checking the identity of a NETCONF-over-TLS server based on their application names:

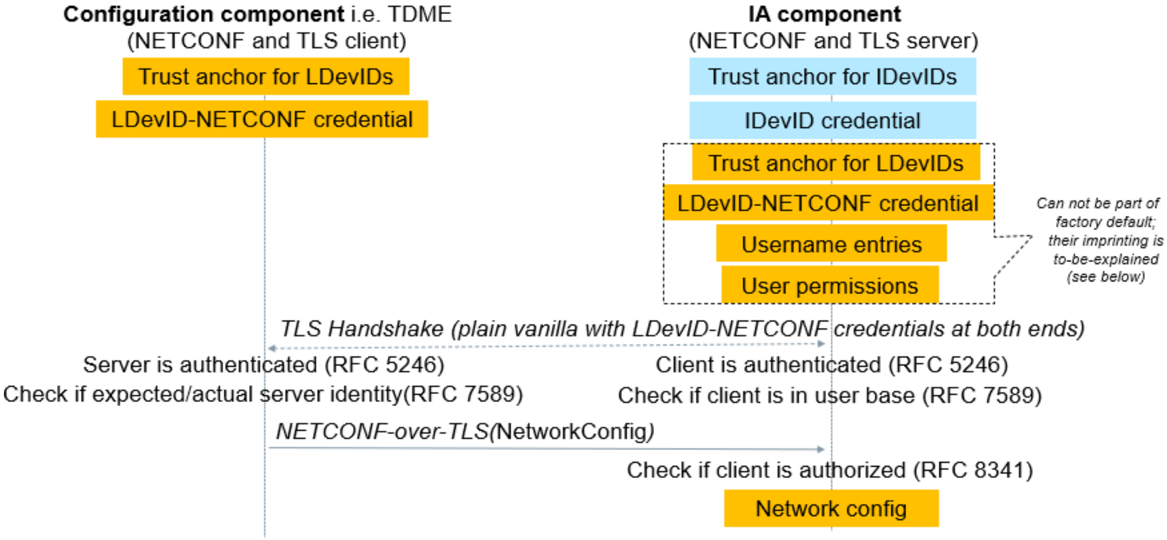
- Actual (FQDN in subjectAltName extension of the EE certificate) is a traditional domain name: case-insensitive ASCII comparison against expected (from address info e.g., request URL)
- Actual (FQDN in subjectAltName extension of the EE certificate) is an internationalized domain name: case-insensitive ASCII comparison against expected (from address info e.g., request URL) after performing any U-label to an A-label, cf. RFC 5890 (see [4]) and RFC 5891 (see [5]) for details.
- Actual (FQDN in subjectAltName extension of the EE certificate) contains a wildcard in its leftmost label:
    - "*" always matches e.g., foo.example.com matches *.example.com (does not match foo.example.net or foo.superexample.com)
    - "<abc>*<xyz>" matches when it matches e.g., foobar.example.com matches foo*.example.com (small-print applies, see RFC 6125)
- Actual (CN in subject field [this is an X.500 DN] of the EE certificate) is a traditional domain name: case-insensitive ASCII comparison against expected (from address info e.g., request URL)

As a *last resort check* (if no FQDN can be found in the subjectAltName extension of the EE certificate) these matching rules can be applied to the CN portion of the subject DN value (small-print applies, see RFC 6125).

Annex C Sequence Charts

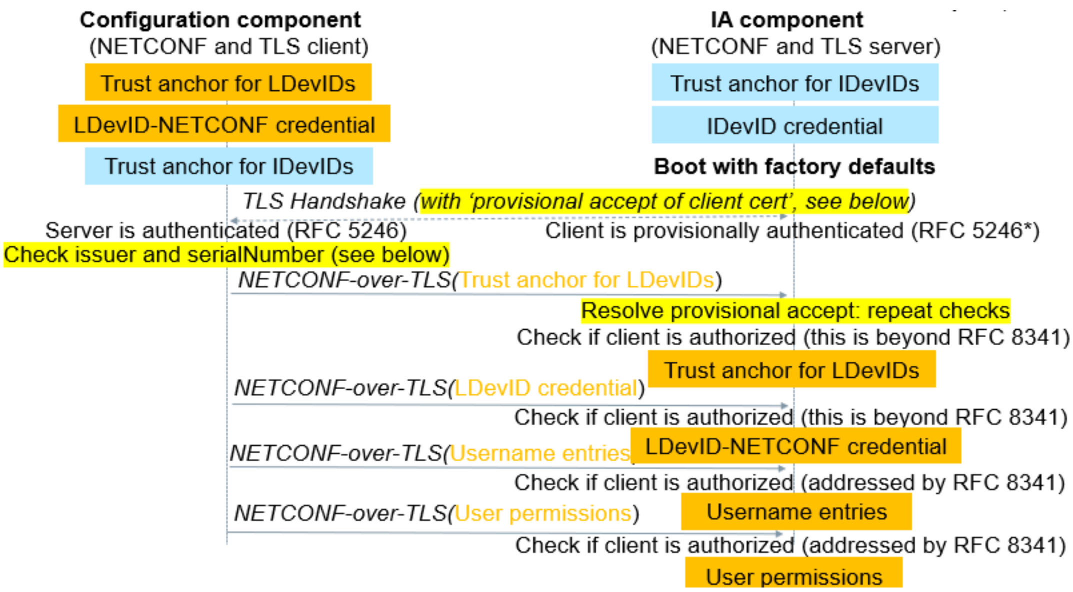
C.1 Post Imprinting Processing Steps

Sequence chart for NETCONF-over-TLS exchanges (RFCs 5246, 7589, 8341) once the IA component was equipped for this purpose:



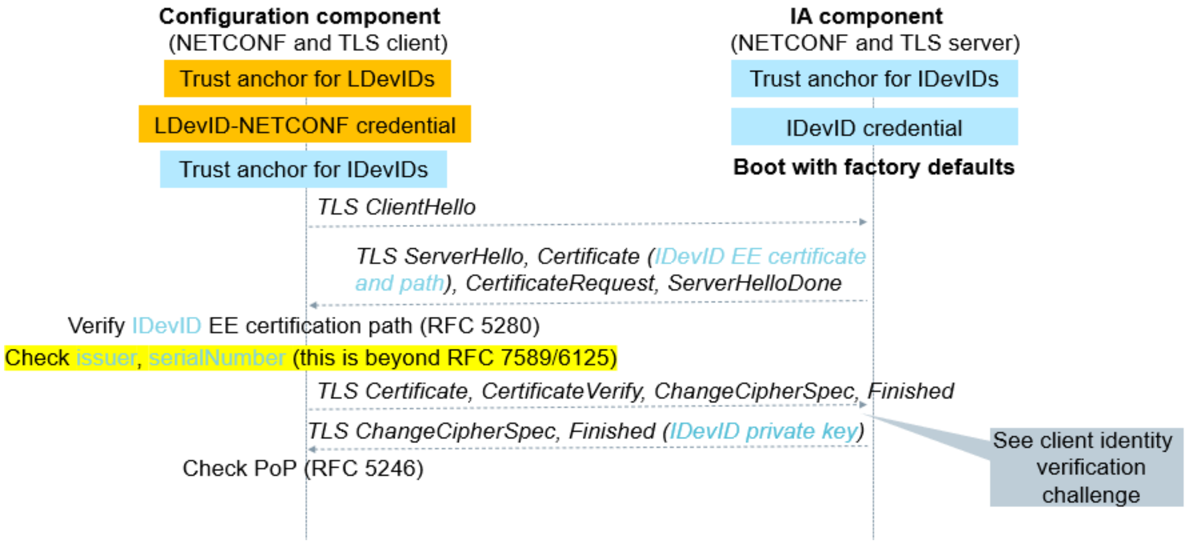
C.2 Imprinting Processing Steps

Sequence chart for equipping an IA component to participate in NETCONF-over-TLS exchanges:



C.2.1 Server Identity Checking Sub-Steps

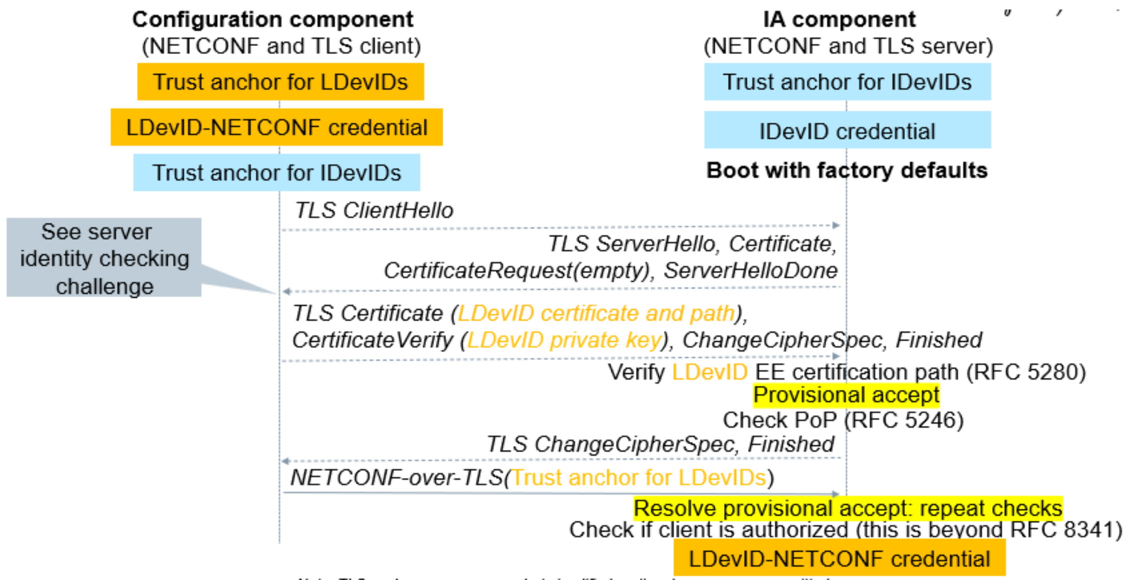
Sequence sub-chart for checking the server identity for NETCONF-over-TLS in case of an IA component that booted in factory default state:



Note: TLS exchanges are somewhat simplified; optional messages are omitted

C.2.2 Client Identity Verification Sub-Steps

Sequence sub-chart for verifying the client identity for NETCONF-over-TLS in case of an IA component that booted in factory default state:



Note: TLS exchanges are somewhat simplified; optional messages are omitted

Annex D TLS Protocol Versions

There are following versions of the TLS protocol:

- TLS 1.0: IETF RFC 2246, January 1999
- TLS 1.1: IETF RFC 4346, April 2006
- TLS 1.2: IETF RFC 5246, August 2008
- TLS 1.3: IETF RFC 8446, August 2018

By the time of writing their fitness assessment is:

- In good standing: TLS 1.2 and 1.3
- Deprecated: TLS 1.0 and 1.1 (see RFC 8996 ‘Deprecating TLS 1.0 and TLS 1.1’, March 2021)

The NETCONF adoption of the TLS protocol versions in good standing is:

- TLS 1.2: used by the current NETCONF-over-TLS standard (RFC 7589)
- TLS 1.3: not used by the current NETCONF-over-TLS standard (RFC 7589). More precisely:
    - By the time of writing there is not yet a NETCONF WG draft document that updates the TLS protocol binding for NETCONF to TLS 1.3 (RFC 8446).
    - There are information model work-in-progress documents (draft-ietf-netconf-tls-client-server-25) that consider an update from {TLS 1.2} to {TLS 1.2, TLS 1.3} in the information model for NETCONF/YANG (not: the NETCONF protocol binding to TLS)