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, ***this version***): equipping IA components with trust anchors and credentials for NETCONF-over-TLS; provides chapter 4.2

iii. Third increment (V0.4, *later*): securely using IA components with NETCONF/YANG; will provide chapter 5

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)

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

ASCII	American Standard Code for Information Interchange
ASN	Abstract Syntax Notation
CA	Certification Authority
CMS	Cryptographic Message Syntax
CN	Common Name (X.500)
CSR	Certificate Signing Request
DER	Distinguished Encoding Rules
DN	Distinguished Name (X.500)
DNS	Domain Name Service
EE	End Entity
FQDN	Fully Qualified Domain Name
HW	HardWare
IA	Industrial Automation
IDevID	Initial Device IDentifier
LDevID	Locally significant Device IDentifier
NETCONF	NETwork CONFiguration
NMDA	Network Management Datastore Architecture
OoB	Out-of-Band
PEM	Privacy Enhanced Mail
PKCS	Public Key Cryptography Standards
SZTP	Secure Zero Touch Provisioning
TDME	TSN Domain Management Entity
TLS	Transport Layer Security
TOFU	Trust On First Use
URL	Uniform Resource Locator
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

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

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]

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 <u>provided</u> 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 <u>not</u> 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:

- 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)?

———————

[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.

[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.

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]

3.2.3 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:

---

[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.

[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"

- 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.

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):

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

a. If authorized: persist the provisioned trust anchor and use it for subsequent certification path validation operations

b. Else: refuse the supplied trust anchor

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])

---

[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.

- `imprintCredential`: imprint a local, deployment-specific credential[15] (LDevID) to an IA component that is booting with factory defaults. Subcases:
  - IA component-external key generation
  - IA component-internal key generation

TODO: `imprintUsernames`, `imprintUserPermissions`, 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)

---

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

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**.

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

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.
  - 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)

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

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>
        <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>
```

[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.

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)

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>
```

[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"

```
                    <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

5 Using the Solution – With Respect To NETCONF/YANG

5.1 Message Exchange Protection for NETCONF/YANG

TODO: describe message exchange protection of NETCONF/YANG exchanges with TLS as secure transport (text is meant to be a profile of RFC 7589; further profiling is needed if further NETCONF secure transports (e.g. SSH, QUIC) shall also be supported by TSN-IA)

TODO: are other secure transports for NETCONF/YANG than TLS in scope of TSN-IA?

5.2 Resource Access Authorization for NETCONF/YANG

TODO: describe resource access authorization for NETCONF/YANG exchanges (text is meant to be a profile of RFC 8341)

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

TODO: describe how the imprinting solution can be exploited to protect other kinds of TSN-IA defined 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[20].
        - 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
- 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):

---

[20] 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

    - 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:

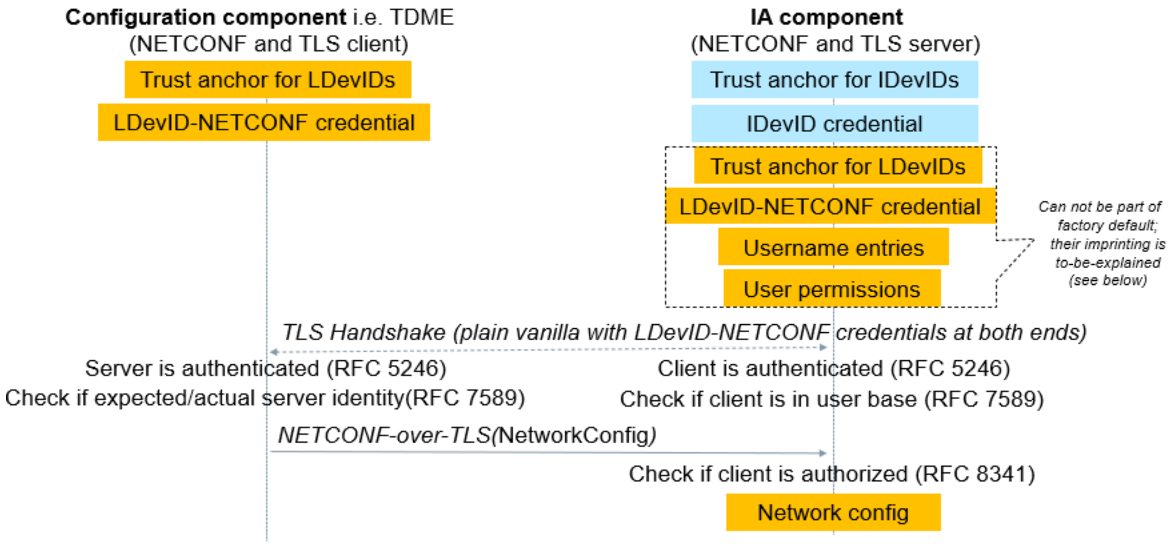
- Actual (FQDN in subjectAltName extension of the EE certificate) is a traditional domain name: case-insensitive ASCII comparison against expected (from address info e.g., request URL)
- Actual (FQDN in subjectAltName extension of the EE certificate) is an internationalized domain name: case-insensitive 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

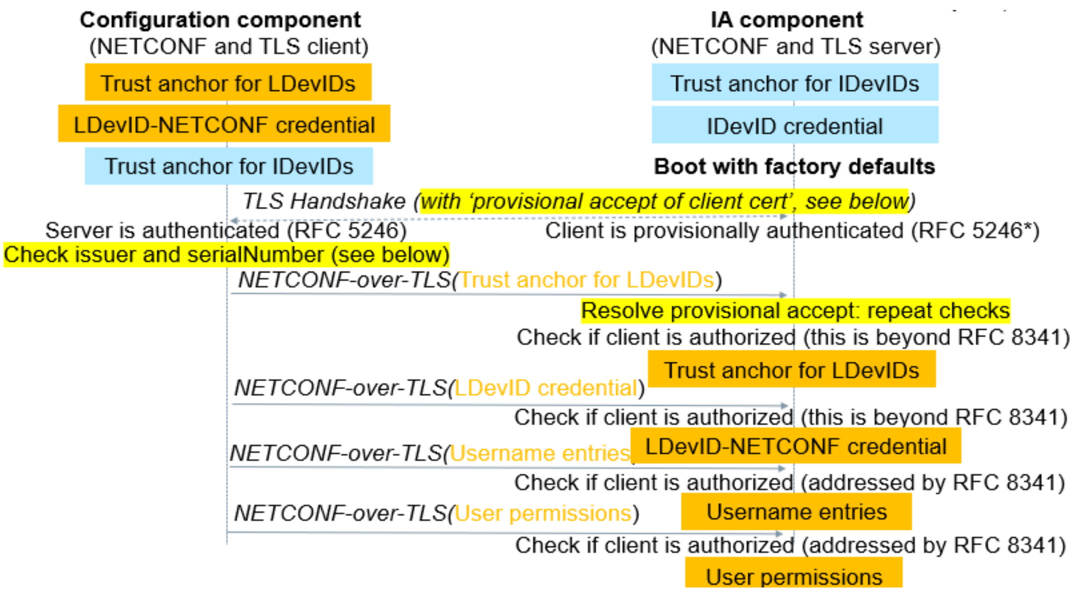
C.1 Post Imprinting Processing Steps

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



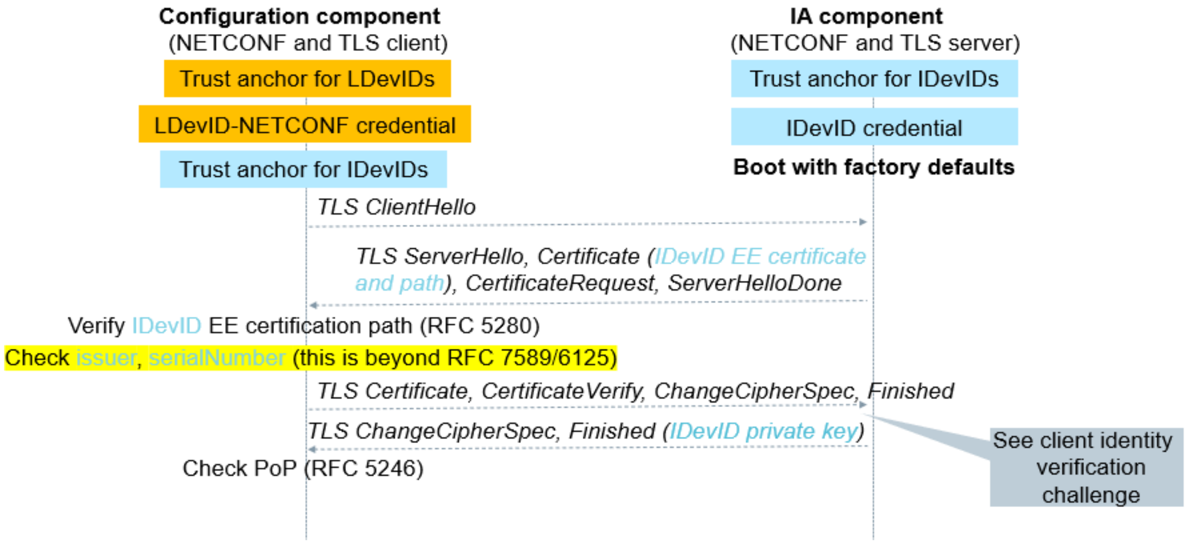
C.2 Imprinting Processing Steps

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



C.2.1 Server Identity Checking Sub-Steps

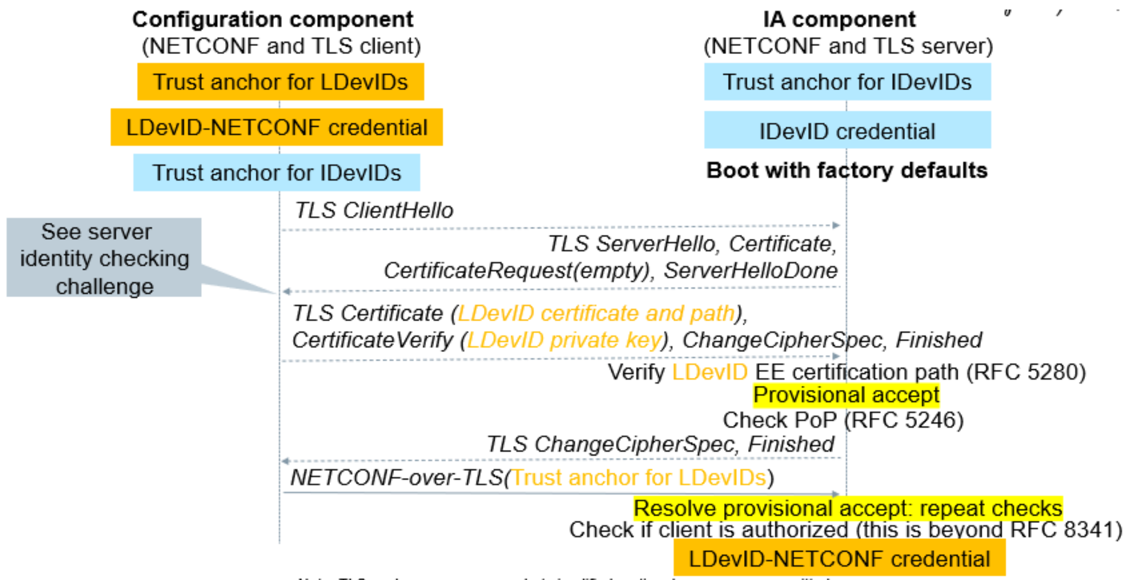
Sequence sub-chart for checking the server identity for NETCONF-over-TLS in case of an IA component that booted in factory default state:



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