Workgroup: NETCONF Working Group
Internet-Draft:
draft-ietf-netconf-tls-client-server-22
Published: 20 August 2020
Intended Status: Standards Track
Expires: 21 February 2021
Authors: K. Watsen
Watsen Networks

YANG Groupings for TLS Clients and TLS Servers

Abstract

This document defines three YANG modules: the first defines groupings for a generic TLS client, the second defines groupings for a generic TLS server, and the third defines common identities and groupings used by both the client and the server. It is intended that these groupings will be used by applications using the TLS protocol.

Editorial Note (To be removed by RFC Editor)

This draft contains placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- *AAAA --> the assigned RFC value for draft-ietf-netconf-crypto-types
- *BBBB --> the assigned RFC value for draft-ietf-netconf-trust-anchors
- *CCCC --> the assigned RFC value for draft-ietf-netconf-keystore
- *DDDD --> the assigned RFC value for draft-ietf-netconf-tcp-client-server
- *FFFF --> the assigned RFC value for this draft

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- *2020-08-20 --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

*Appendix A. Change Log

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 February 2021.

Copyright Notice

Table of Contents

1. Introduction

This document defines three YANG 1.1 [RFC7950] modules: the first defines a grouping for a generic TLS client, the second defines a grouping for a generic TLS server, and the third defines identities and groupings common to both the client and the server (TLS is defined in [RFC5246]). It is intended that these groupings will be

used by applications using the TLS protocol. For instance, these groupings could be used to help define the data model for an HTTPS [RFC2818] server or a NETCONF over TLS [RFC7589] based server.

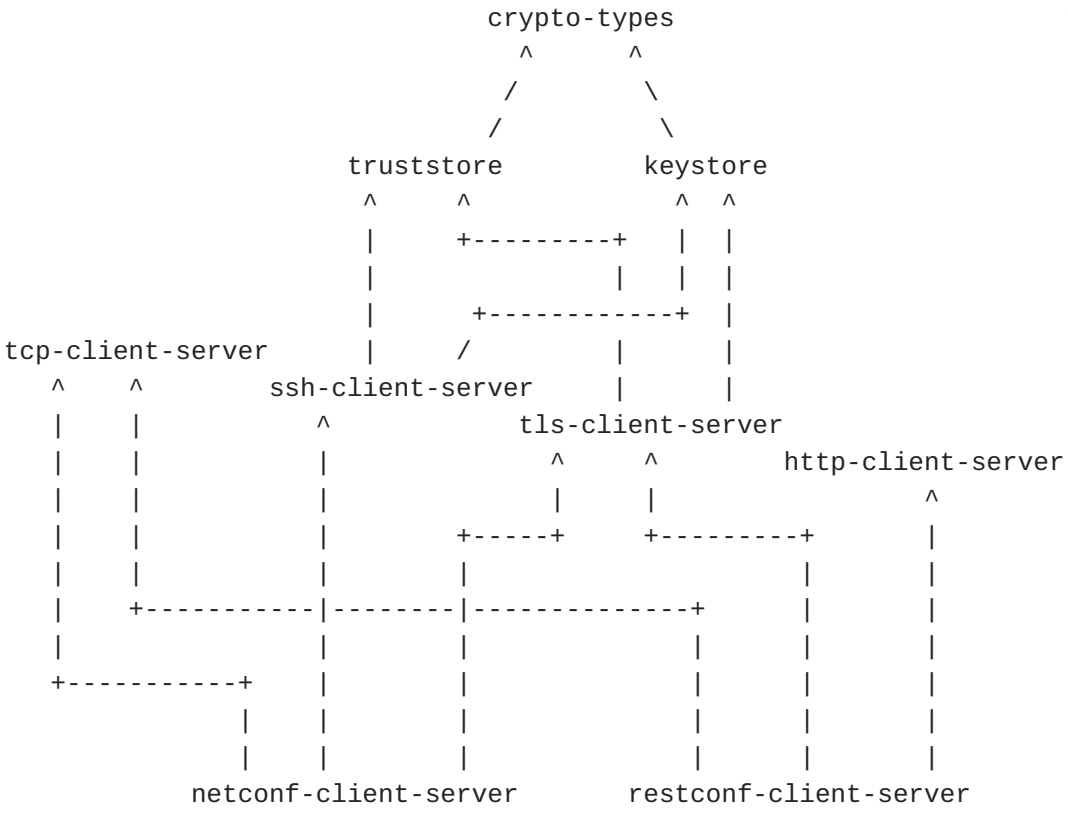
The client and server YANG modules in this document each define one grouping, which is focused on just TLS-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen-on or connect-to. This affords applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the "ssh-server-grouping" grouping for the TLS parts it provides, while adding data nodes for the TCP-level call-home configuration.

1.1. Relation to other RFCs

This document presents one or more YANG modules [RFC7950] that are part of a collection of RFCs that work together to define configuration modules for clients and servers of both the NETCONF [RFC6241] and RESTCONF [RFC8040] protocols.

The modules have been defined in a modular fashion to enable their use by other efforts, some of which are known to be in progress at the time of this writing, with many more expected to be defined in time.

The normative dependency relationship between the various RFCs in the collection is presented in the below diagram. The labels in the diagram represent the primary purpose provided by each RFC. Hyperlinks to each RFC are provided below the diagram.



Label in Diagram	Originating RFC
crypto-types	[I-D.ietf-netconf-crypto-types]
truststore	[I-D.ietf-netconf-trust-anchors]
keystore	[I-D.ietf-netconf-keystore]
tcp-client-server	[I-D.ietf-netconf-tcp-client-server]
ssh-client-server	[I-D.ietf-netconf-ssh-client-server]
tls-client-server	[I-D.ietf-netconf-tls-client-server]
http-client-server	[I-D.ietf-netconf-http-client-server]
netconf-client-server	[I-D.ietf-netconf-netconf-client-server]
restconf-client-server	[I-D.ietf-netconf-restconf-client-server]

Table 1: Label to RFC Mapping

1.2. Specification Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3. Adherence to the NMDA

This document in compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, as described in [I-

D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore], trust anchors and keys installed during manufacturing are expected to appear in <operational>.

2. The "ietf-tls-common" Module

The TLS common model presented in this section contains identities and groupings common to both TLS clients and TLS servers. The "hello-params-grouping" grouping can be used to configure the list of TLS algorithms permitted by the TLS client or TLS server. The lists of algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the TLS transport layer connection. The ability to restrict the algorithms allowed is provided in this grouping for TLS clients and TLS servers that are capable of doing so and may serve to make TLS clients and TLS servers compliant with local security policies. This model supports both TLS1.2 [RFC5246] and TLS 1.3 [RFC8446].

TLS 1.2 and TLS 1.3 have different ways defining their own supported cryptographic algorithms, see TLS and DTLS IANA registries page (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml):

*TLS 1.2 defines four categories of registries for cryptographic algorithms: TLS Cipher Suites, TLS SignatureAlgorithm, TLS HashAlgorithm, TLS Supported Groups. TLS Cipher Suites plays the role of combining all of them into one set, as each value of the set represents a unique and feasible combination of all the cryptographic algorithms, and thus the other three registry categories do not need to be considered here. In this document, the TLS common model only chooses those TLS1.2 algorithms in TLS Cipher Suites which are marked as recommended: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_PSK_WITH_AES_128_GCM_SHA256, TLS_DHE_PSK_WITH_AES_256_GCM_SHA384, and so on. All chosen algorithms are enumerated in Table 1-1 below;

*TLS 1.3 defines its supported algorithms differently. Firstly, it defines three categories of registries for cryptographic algorithms: TLS Cipher Suites, TLS SignatureScheme, TLS Supported Groups. Secondly, all three of these categories are useful, since they represent different parts of all the supported algorithms respectively. Thus, all of these registries categories are considered here. In this draft, the TLS common model chooses only those TLS1.3 algorithms specified in B.4, 4.2.3, 4.2.7 of [RFC8446].

Thus, in order to support both TLS1.2 and TLS1.3, the cipher-suites part of the "hello-params-grouping" grouping should include three parameters for configuring its permitted TLS algorithms, which are: TLS Cipher Suites, TLS SignatureScheme, TLS Supported Groups. Note that TLS1.2 only uses TLS Cipher Suites.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive.

2.1. Data Model Overview

This section provides an overview of the "ietf-tls-common" module in terms of its features, identitiesm and groupings.

2.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-tls-common" module:

```
Features:
  +-- tls-1_0
  +-- tls-1_1
  +-- tls-1_2
  +-- tls-1_3
  +-- tls-ecc
  +-- tls-dhe
  +-- tls-3des
  +-- tls-gcm
  +-- tls-sha2
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

2.1.2. Identities

The following diagram illustrates the relationship amongst the "identity" statements defined in the "ietf-tls-common" module:

```
Identities:
  +-- tls-version-base
  |  +-- tls-1.0
  |  +-- tls-1.1
  |  +-- tls-1.2
  +-- cipher-suite-base
     +-- rsa-with-aes-128-cbc-sha
     +-- rsa-with-aes-256-cbc-sha
     +-- rsa-with-aes-128-cbc-sha256
     +-- rsa-with-aes-256-cbc-sha256
     +-- dhe-rsa-with-aes-128-cbc-sha
     +-- dhe-rsa-with-aes-256-cbc-sha
     +-- dhe-rsa-with-aes-128-cbc-sha256
     +-- dhe-rsa-with-aes-256-cbc-sha256
     +-- ecdhe-ecdsa-with-aes-128-cbc-sha256
     +-- ecdhe-ecdsa-with-aes-256-cbc-sha384
     +-- ecdhe-rsa-with-aes-128-cbc-sha256
     +-- ecdhe-rsa-with-aes-256-cbc-sha384
     +-- ecdhe-ecdsa-with-aes-128-gcm-sha256
     +-- ecdhe-ecdsa-with-aes-256-gcm-sha384
     +-- ecdhe-rsa-with-aes-128-gcm-sha256
     +-- ecdhe-rsa-with-aes-256-gcm-sha384
     +-- rsa-with-3des-ede-cbc-sha
     +-- ecdhe-rsa-with-3des-ede-cbc-sha
     +-- ecdhe-rsa-with-aes-128-cbc-sha
     +-- ecdhe-rsa-with-aes-256-cbc-sha
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Comments:

- The diagram shows that there are two base identities.
- One base identity is used to specific TLS versions, while the other is used to specify cipher-suites.
- These base identities are "abstract", in the object orientied programming sense, in that they only define a "class" of things, rather than a specific thing.

2.1.3. Groupings

The following diagram lists all the "grouping" statements defined in the "ietf-tls-common" module:

```
Groupings:
  +-- hello-params-grouping
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Each of these groupings are presented in the following subsections.

2.1.3.1. The "hello-params-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "hello-params-grouping" grouping:

```
grouping hello-params-grouping
  +-- tls-versions
  |  +-- tls-version*   identityref
  +-- cipher-suites
     +-- cipher-suite*   identityref
```

Comments:

*This grouping is used by both the "tls-client-grouping" and the "tls-server-grouping" groupings defined in Section 3.1.2.1 and Section 4.1.2.1, respectively.

*This grouping enables client and server configurations to specify the TLS versions and cipher suites that are to be used when establishing TLS sessions.

*The "cipher-suites" list is "ordered-by user".

2.1.4. Protocol-accessible Nodes

The "ietf-tls-common" module does not contain any protocol-accessible nodes, but the module needs to be "implemented", as described in Section 5.6.5 of [RFC7950], in order for the identities in Section 2.1.2 to be defined.

2.2. Example Usage

This section shows how it would appear if the "hello-params-grouping" grouping were populated with some data.

```
<hello-params
   xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-common"
   xmlns:tlscmn="urn:ietf:params:xml:ns:yang:ietf-tls-common">
  <tls-versions>
    <tls-version>tlscmn:tls-1.1</tls-version>
    <tls-version>tlscmn:tls-1.2</tls-version>
  </tls-versions>
  <cipher-suites>
    <cipher-suite>tlscmn:dhe-rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>tlscmn:rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>tlscmn:rsa-with-3des-ede-cbc-sha</cipher-suite>
  </cipher-suites>
</hello-params>
```

2.3. YANG Module

This YANG module has a normative references to [RFC4346], [RFC5246], [RFC5288], [RFC5289], and [RFC8422].

This YANG module has a informative references to [RFC2246], [RFC4346], [RFC5246], and [RFC8446].

<CODE BEGINS> file "ietf-tls-common@2020-08-20.yang"

```
module ietf-tls-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-common";
  prefix tlscmn;

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kent+ietf@watsen.net>
     Author:   Gary Wu <mailto:garywu@cisco.com>";

   description
    "This module defines a common features, identities, and
     groupings for Transport Layer Security (TLS).

     Copyright (c) 2020 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Simplified
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

     This version of this YANG module is part of RFC FFFF
     (https://www.rfc-editor.org/info/rfcFFFF); see the RFC
     itself for full legal notices.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
     'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
     are to be interpreted as described in BCP 14 (RFC 2119)
     (RFC 8174) when, and only when, they appear in all
     capitals, as shown here.";

  revision 2020-08-20 {
    description
      "Initial version";
    reference
      "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
  }

  // Features

  feature tls-1_0 {
    description
```

```
    "TLS Protocol Version 1.0 is supported.";
  reference
    "RFC 2246: The TLS Protocol Version 1.0";
}

feature tls-1_1 {
  description
    "TLS Protocol Version 1.1 is supported.";
  reference
    "RFC 4346: The Transport Layer Security (TLS) Protocol
               Version 1.1";
}

feature tls-1_2 {
  description
    "TLS Protocol Version 1.2 is supported.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

feature tls-1_3 {
  description
    "TLS Protocol Version 1.2 is supported.";
  reference
    "RFC 8446: The Transport Layer Security (TLS) Protocol
               Version 1.3";
}

feature tls-ecc {
  description
    "Elliptic Curve Cryptography (ECC) is supported for TLS.";
  reference
    "RFC 8422: Elliptic Curve Cryptography (ECC) Cipher Suites
               for Transport Layer Security (TLS)";
}

feature tls-dhe {
  description
    "Ephemeral Diffie-Hellman key exchange is supported for TLS.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

feature tls-3des {
  description
    "The Triple-DES block cipher is supported for TLS.";
  reference
```

```
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

feature tls-gcm {
  description
    "The Galois/Counter Mode authenticated encryption mode is
     supported for TLS.";
  reference
    "RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for
               TLS";
}

feature tls-sha2 {
  description
    "The SHA2 family of cryptographic hash functions is supported
     for TLS.";
  reference
    "FIPS PUB 180-4: Secure Hash Standard (SHS)";
}

// Identities

identity tls-version-base {
  description
    "Base identity used to identify TLS protocol versions.";
}

identity tls-1.0 {
  if-feature "tls-1_0";
  base tls-version-base;
  description
    "TLS Protocol Version 1.0.";
  reference
    "RFC 2246: The TLS Protocol Version 1.0";
}

identity tls-1.1 {
  if-feature "tls-1_1";
  base tls-version-base;
  description
    "TLS Protocol Version 1.1.";
  reference
    "RFC 4346: The Transport Layer Security (TLS) Protocol
               Version 1.1";
}

identity tls-1.2 {
  if-feature "tls-1_2";
```

```
  base tls-version-base;
  description
    "TLS Protocol Version 1.2.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity cipher-suite-base {
  description
    "Base identity used to identify TLS cipher suites.";
}

identity rsa-with-aes-128-cbc-sha {
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity rsa-with-aes-256-cbc-sha {
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity rsa-with-aes-128-cbc-sha256 {
  if-feature "tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA256.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity rsa-with-aes-256-cbc-sha256 {
  if-feature "tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA256.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
```

```
}

identity dhe-rsa-with-aes-128-cbc-sha {
  if-feature "tls-dhe";
  base cipher-suite-base;
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity dhe-rsa-with-aes-256-cbc-sha {
  if-feature "tls-dhe";
  base cipher-suite-base;
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity dhe-rsa-with-aes-128-cbc-sha256 {
  if-feature "tls-dhe and tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA256.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity dhe-rsa-with-aes-256-cbc-sha256 {
  if-feature "tls-dhe and tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA256.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity ecdhe-ecdsa-with-aes-128-cbc-sha256 {
  if-feature "tls-ecc and tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
```

```
              SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-ecdsa-with-aes-256-cbc-sha384 {
    if-feature "tls-ecc and tls-sha2";
    base cipher-suite-base;
    description
      "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
              SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-rsa-with-aes-128-cbc-sha256 {
    if-feature "tls-ecc and tls-sha2";
    base cipher-suite-base;
    description
      "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
              SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-rsa-with-aes-256-cbc-sha384 {
    if-feature "tls-ecc and tls-sha2";
    base cipher-suite-base;
    description
      "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
              SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-ecdsa-with-aes-128-gcm-sha256 {
    if-feature "tls-ecc and tls-gcm and tls-sha2";
    base cipher-suite-base;
    description
      "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
              SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-ecdsa-with-aes-256-gcm-sha384 {
    if-feature "tls-ecc and tls-gcm and tls-sha2";
    base cipher-suite-base;
    description
      "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.";
    reference
```

```
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
               SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-rsa-with-aes-128-gcm-sha256 {
  if-feature "tls-ecc and tls-gcm and tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
               SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-rsa-with-aes-256-gcm-sha384 {
  if-feature "tls-ecc and tls-gcm and tls-sha2";
  base cipher-suite-base;
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
               SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity rsa-with-3des-ede-cbc-sha {
  if-feature "tls-3des";
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_3DES_EDE_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
}

identity ecdhe-rsa-with-3des-ede-cbc-sha {
  if-feature "tls-ecc and tls-3des";
  base cipher-suite-base;
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA.";
  reference
    "RFC 8422: Elliptic Curve Cryptography (ECC) Cipher Suites
               for Transport Layer Security (TLS)";
}

identity ecdhe-rsa-with-aes-128-cbc-sha {
  if-feature "tls-ecc";
  base cipher-suite-base;
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA.";
```

```
  reference
    "RFC 8422: Elliptic Curve Cryptography (ECC) Cipher Suites
               for Transport Layer Security (TLS)";
}

identity ecdhe-rsa-with-aes-256-cbc-sha {
  if-feature "tls-ecc";
  base cipher-suite-base;
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA.";
  reference
    "RFC 8422: Elliptic Curve Cryptography (ECC) Cipher Suites
               for Transport Layer Security (TLS)";
}

// Groupings

grouping hello-params-grouping {
  description
    "A reusable grouping for TLS hello message parameters.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
               Version 1.2";
  container tls-versions {
    description
      "Parameters regarding TLS versions.";
    leaf-list tls-version {
      type identityref {
        base tls-version-base;
      }
      description
        "Acceptable TLS protocol versions.

         If this leaf-list is not configured (has zero elements)
         the acceptable TLS protocol versions are implementation-
         defined.";
    }
  }
  container cipher-suites {
    description
      "Parameters regarding cipher suites.";
    leaf-list cipher-suite {
      type identityref {
        base cipher-suite-base;
      }
      ordered-by user;
      description
        "Acceptable cipher suites in order of descending
         preference.  The configured host key algorithms should
```

```
          be compatible with the algorithm used by the configured
          private key.  Please see Section 5 of RFC FFFF for
          valid combinations.

          If this leaf-list is not configured (has zero elements)
          the acceptable cipher suites are implementation-
          defined.";
        reference
          "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
      }
    }
  }
}
```

<CODE ENDS>

3. The "ietf-tls-client" Module

This section defines a YANG 1.1 [RFC7950] module called "ietf-tls-client". A high-level overview of the module is provided in Section 3.1. Examples illustatrating the module's use are provided in Examples (Section 3.2). The YANG module itself is defined in Section 3.3.

3.1. Data Model Overview

This section provides an overview of the "ietf-tls-client" module in terms of its features and groupings.

3.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-tls-client" module:

```
Features:
  +-- tls-client-hello-params-config
  +-- tls-client-keepalives
  +-- x509-certificate-auth
  +-- raw-public-key-auth
  +-- psk-auth
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

3.1.2. Groupings

The following diagram lists all the "grouping" statements defined in the "ietf-tls-client" module:

```
Groupings:
  +-- tls-client-grouping
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Each of these groupings are presented in the following subsections.

3.1.2.1. The "tls-client-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "tls-client-grouping" grouping:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

  grouping tls-client-grouping
    +-- client-identity!
    |  +-- (auth-type)
    |     +--:(certificate) {x509-certificate-auth}?
    |     |  +-- certificate
    |     |     +---u ks:local-or-keystore-end-entity-cert-with-key-\
grouping
    |     +--:(raw-public-key) {raw-public-key-auth}?
    |     |  +-- raw-private-key
    |     |     +---u ks:local-or-keystore-asymmetric-key-grouping
    |     +--:(psk) {psk-auth}?
    |        +-- psk
    |           +---u ks:local-or-keystore-symmetric-key-grouping
    |           +-- id?
    |                   string
    +-- server-authentication
    |  +-- ca-certs! {x509-certificate-auth}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- ee-certs! {x509-certificate-auth}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- raw-public-keys! {raw-public-key-auth}?
    |  |  +---u ts:local-or-truststore-public-keys-grouping
    |  +-- psks?              empty {psk-auth}?
    +-- hello-params {tls-client-hello-params-config}?
    |  +---u tlscmn:hello-params-grouping
    +-- keepalives {tls-client-keepalives}?
       +-- peer-allowed-to-send?   empty
       +-- test-peer-aliveness!
          +-- max-wait?       uint16
          +-- max-attempts?   uint8
```

Comments:

*The "client-identity" node, which is optionally configured (as client authentication MAY occur at a higher protocol layer), configures identity credentials, each enabled by a "feature" statement defined in Section 3.1.1.

*The "server-authentication" node configures trust anchors for authenticating the TLS server, with each option enabled by a "feature" statement.

*The "hello-params" node , which must be enabled by a feature, configures parameters for the TLS sessions established by this configuration.

*The "keepalives" node, which must be enabled by a feature, configures a "presence" container for testing the aliveness of the TLS server. The aliveness-test occurs at the TLS protocol layer.

*For the referenced grouping statement(s):

- The "local-or-keystore-end-entity-cert-with-key-grouping" grouping is discussed in Section 2.1.3.6 of [I-D.ietf-netconf-keystore].
- The "local-or-keystore-asymmetric-key-grouping" grouping is discussed in Section 2.1.3.4 of [I-D.ietf-netconf-keystore].
- The "local-or-keystore-symmetric-key-grouping" grouping is discussed in Section 2.1.3.3 of [I-D.ietf-netconf-keystore].
- The "local-or-truststore-certs-grouping" grouping is discussed in Section 2.1.3.1 of [I-D.ietf-netconf-trust-anchors].
- The "local-or-truststore-public-keys-grouping" grouping is discussed in Section 2.1.3.2 of [I-D.ietf-netconf-trust-anchors].
- The "hello-params-grouping" grouping is discussed in Section 2.1.3.1 in this document.

3.1.3. Protocol-accessible Nodes

The "ietf-tls-client" module does not contain any protocol-accessible nodes.

3.2. Example Usage

This section presents two examples showing the "tls-client-grouping" grouping populated with some data. These examples are effectively the same except the first configures the client identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 2 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following configuration example uses local-definitions for the client identity and server authentication:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<tls-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-client"
  xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <certificate>
      <local-definition>
        <public-key-format>ct:subject-public-key-info-format</public\
-key-format>
        <public-key>base64encodedvalue==</public-key>
        <private-key-format>ct:rsa-private-key-format</private-key-f\
ormat>
        <cleartext-private-key>base64encodedvalue==</cleartext-priva\
te-key>
        <cert-data>base64encodedvalue==</cert-data>
      </local-definition>
    </certificate>
    <!-- TESTED, BUT COMMENTED OUT DUE TO ONLY ONE ALLOWED AT A TIME
    <raw-private-key>
      <local-definition>
        <public-key-format>ct:subject-public-key-info-format</public\
-key-format>
        <public-key>base64encodedvalue==</public-key>
        <private-key-format>ct:rsa-private-key-format</private-key-f\
ormat>
        <cleartext-private-key>base64encodedvalue==</cleartext-priva\
te-key>
      </local-definition>
    </raw-private-key>
    <psk>
      <local-definition>
        <key-format>ct:octet-string-key-format</key-format>
        <cleartext-key>base64encodedvalue==</cleartext-key>
      </local-definition>
    </psk>
    -->
  </client-identity>

  <!-- which certificates will this client trust -->
  <server-authentication>
    <ca-certs>
      <local-definition>
        <certificate>
          <name>Server Cert Issuer #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
```

```
        <certificate>
          <name>Server Cert Issuer #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ca-certs>
    <ee-certs>
      <local-definition>
        <certificate>
          <name>My Application #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>My Application #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ee-certs>
    <raw-public-keys>
      <local-definition>
        <public-key>
          <name>corp-fw1</name>
          <public-key-format>ct:subject-public-key-info-format</publ\
ic-key-format>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
        <public-key>
          <name>corp-fw1</name>
          <public-key-format>ct:subject-public-key-info-format</publ\
ic-key-format>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
      </local-definition>
    </raw-public-keys>
    <psks/>
  </server-authentication>

  <keepalives>
    <test-peer-aliveness>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </test-peer-aliveness>
  </keepalives>

</tls-client>
```

The following configuration example uses keystore-references for the client identity and truststore-references for server authentication: from the keystore:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<tls-client xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-client">

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <certificate>
      <keystore-reference>
        <asymmetric-key>rsa-asymmetric-key</asymmetric-key>
        <certificate>ex-rsa-cert</certificate>
      </keystore-reference>
    </certificate>
    <!-- TESTED, BUT COMMENTED OUT DUE TO ONLY ONE ALLOWED AT A TIME
    <raw-private-key>
      <keystore-reference>raw-private-key</keystore-reference>
    </raw-private-key>
    <psk>
      <keystore-reference>encrypted-symmetric-key</keystore-referenc\
e>
    </psk>
    -->
  </client-identity>

  <!-- which certificates will this client trust -->
  <server-authentication>
    <ca-certs>
      <truststore-reference>trusted-server-ca-certs</truststore-refe\
rence>
    </ca-certs>
    <ee-certs>
      <truststore-reference>trusted-server-ee-certs</truststore-refe\
rence>
    </ee-certs>
    <raw-public-keys>
      <truststore-reference>Raw Public Keys for TLS Servers</truststo\
ore-reference>
    </raw-public-keys>
    <psks/>
  </server-authentication>

  <keepalives>
    <test-peer-aliveness>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </test-peer-aliveness>
  </keepalives>

</tls-client>
```

3.3. YANG Module

This YANG module has normative references to [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-tls-client@2020-08-20.yang"

```
module ietf-tls-client {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-client";
  prefix tlsc;

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC AAAA: YANG Data Types and Groupings for Cryptography";
  }

  import ietf-truststore {
    prefix ts;
    reference
      "RFC BBBB: A YANG Data Model for a Truststore";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC CCCC: A YANG Data Model for a Keystore";
  }

  import ietf-tls-common {
    prefix tlscmn;
    revision-date 2020-08-20; // stable grouping definitions
    reference
      "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kent+ietf@watsen.net>
     Author:   Gary Wu <mailto:garywu@cisco.com>";

  description
    "This module defines reusable groupings for TLS clients that
     can be used as a basis for specific TLS client instances.

     Copyright (c) 2020 IETF Trust and the persons identified
```

```
   as authors of the code. All rights reserved.

   Redistribution and use in source and binary forms, with
   or without modification, is permitted pursuant to, and
   subject to the license terms contained in, the Simplified
   BSD License set forth in Section 4.c of the IETF Trust's
   Legal Provisions Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC FFFF
   (https://www.rfc-editor.org/info/rfcFFFF); see the RFC
   itself for full legal notices.

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
   'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
   'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
   are to be interpreted as described in BCP 14 (RFC 2119)
   (RFC 8174) when, and only when, they appear in all
   capitals, as shown here.";

 revision 2020-08-20 {
   description
     "Initial version";
   reference
     "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
 }

 // Features

 feature tls-client-hello-params-config {
   description
     "TLS hello message parameters are configurable on a TLS
      client.";
 }

 feature tls-client-keepalives {
   description
     "Per socket TLS keepalive parameters are configurable for
      TLS clients on the server implementing this feature.";
 }

 feature x509-certificate-auth {
   description
     "Indicates that the client supports authenticating servers
      using X.509 certificates.";
 }

 feature raw-public-key-auth {
   description
     "Indicates that the client supports authenticating servers
```

```
     using ray public keys.";
}

feature psk-auth {
  description
    "Indicates that the client supports authenticating servers
     using PSKs (pre-shared or pairwise-symmetric keys).";
}



// Groupings

grouping tls-client-grouping {
  description
    "A reusable grouping for configuring a TLS client without
     any consideration for how an underlying TCP session is
     established.

     Note that this grouping uses fairly typical descendent
     node names such that a stack of 'uses' statements will
     have name conflicts.  It is intended that the consuming
     data model will resolve the issue (e.g., by wrapping
     the 'uses' statement in a container called
     'tls-client-parameters').  This model purposely does
     not do this itself so as to provide maximum flexibility
     to consuming models.";

  container client-identity {
    nacm:default-deny-write;
    presence
      "Indicates that TLS-level client authentication
       is sent.  Present so that the 'choice' node's
       mandatory true doesn't imply that a client
       identity must be configured.";
    description
      "Identity credentials the TLS client MAY present when
       establishing a connection to a TLS server.  If not
       configured, then client authentication is presumed to
       occur a protocol layer above TLS.  When configured,
       and requested by the TLS server when establishing a
       TLS session, these credentials are passed in the
       Certificate message defined in Section 7.4.2 of
       RFC 5246.";
    reference
      "RFC 5246: The Transport Layer Security (TLS) Protocol
                 Version 1.2
       RFC CCCC: A YANG Data Model for a Keystore";
    choice auth-type {
```

```
mandatory true;
description
  "A choice amongst available authentication types.";
case certificate {
  if-feature x509-certificate-auth;
  container certificate {
    description
      "Specifies the client identity using a certificate.";
    uses
      ks:local-or-keystore-end-entity-cert-with-key-grouping{
      refine "local-or-keystore/local/local-definition" {
        must 'public-key-format'
             + ' = "ct:subject-public-key-info-format"';
      }
      refine "local-or-keystore/keystore/keystore-reference"
             + "/asymmetric-key" {
        must 'deref(.)/../ks:public-key-format'
             + ' = "ct:subject-public-key-info-format"';
      }
    }
  }
}
case raw-public-key {
  if-feature raw-public-key-auth;
  container raw-private-key {
    description
      "Specifies the client identity using a raw
       private key.";
    uses ks:local-or-keystore-asymmetric-key-grouping {
      refine "local-or-keystore/local/local-definition" {
        must 'public-key-format'
             + ' = "ct:subject-public-key-info-format"';
      }
      refine "local-or-keystore/keystore"
             + "/keystore-reference" {
        must 'deref(.)/../ks:public-key-format'
             + ' = "ct:subject-public-key-info-format"';
      }
    }
  }
}
case psk {
  if-feature psk-auth;
  container psk {
    description
      "Specifies the client identity using a PSK (pre-shared
      or pairwise-symmetric key).";
    uses ks:local-or-keystore-symmetric-key-grouping;
    leaf id {
```

```
          type string;
          description
            "The key 'psk_identity' value used in the TLS
             'ClientKeyExchange' message.";
          reference
            "RFC 4279: Pre-Shared Key Ciphersuites for
                       Transport Layer Security (TLS)";
        }
      }
    }
  }
} // container client-identity

container server-authentication {
  nacm:default-deny-write;
  must 'ca-certs or ee-certs or raw-public-keys or psks';
  description
    "Specifies how the TLS client can authenticate TLS servers.
     Any combination of credentials is additive and unordered.

     Note that no configuration is required for PSK (pre-shared
     or pairwise-symmetric key) based authentication as the key
     is necessarily the same as configured in the '../client-
     identity' node.";
  container ca-certs {
    if-feature "x509-certificate-auth";
    presence
      "Indicates that the TLS client can authenticate TLS servers
       using configured certificate authority certificates.";
    description
      "A set of certificate authority (CA) certificates used by
       the TLS client to authenticate TLS server certificates.
       A server certificate is authenticated if it has a valid
       chain of trust to a configured CA certificate.";
    reference
      "RFC BBBB: A YANG Data Model for a Truststore";
    uses ts:local-or-truststore-certs-grouping;
  }
  container ee-certs {
    if-feature "x509-certificate-auth";
    presence
      "Indicates that the TLS client can authenticate TLS
       servers using configured server certificates.";
    description
      "A set of server certificates (i.e., end entity
       certificates) used by the TLS client to authenticate
       certificates presented by TLS servers.  A server
       certificate is authenticated if it is an exact
       match to a configured server certificate.";
```

```
      reference
        "RFC BBBB: A YANG Data Model for a Truststore";
      uses ts:local-or-truststore-certs-grouping;
    }
    container raw-public-keys {
      if-feature "raw-public-key-auth";
      presence
        "Indicates that the TLS client can authenticate TLS
         servers using configured server certificates.";
      description
        "A set of raw public keys used by the TLS client to
         authenticate raw public keys presented by the TLS
         server.  A raw public key is authenticated if it
         is an exact match to a configured raw public key.";
      reference
        "RFC BBBB: A YANG Data Model for a Truststore";
      uses ts:local-or-truststore-public-keys-grouping {
        refine "local-or-truststore/local/local-definition"
               + "/public-key" {
          must 'public-key-format'
               + ' = "ct:subject-public-key-info-format"';
        }
        refine "local-or-truststore/truststore"
               + "/truststore-reference" {
          must 'deref(.)/../*/ts:public-key-format'
               + ' = "ct:subject-public-key-info-format"';
        }
      }
    }
    leaf psks {
      if-feature "psk-auth";
      type empty;
      description
        "Indicates that the TLS client can authenticate TLS servers
         using configure PSKs (pre-shared or pairwise-symmetric
         keys).

         No configuration is required since the PSK value is the
         same as PSK value configured in the 'client-identity'
         node.";
    }
  } // container server-authentication

  container hello-params {
    nacm:default-deny-write;
    if-feature "tls-client-hello-params-config";
    uses tlscmn:hello-params-grouping;
    description
      "Configurable parameters for the TLS hello message.";
```

```
} // container hello-params

container keepalives {
  nacm:default-deny-write;
  if-feature "tls-client-keepalives";
  description
    "Configures the keepalive policy for the TLS client.";
  leaf peer-allowed-to-send {
    type empty;
    description
      "Indicates that the remote TLS server is allowed to send
       HeartbeatRequest messages, as defined by RFC 6520
       to this TLS client.";
    reference
      "RFC 6520: Transport Layer Security (TLS) and Datagram
       Transport Layer Security (DTLS) Heartbeat Extension";
  }
  container test-peer-aliveness {
    presence
      "Indicates that the TLS client proactively tests the
       aliveness of the remote TLS server.";
    description
      "Configures the keep-alive policy to proactively test
       the aliveness of the TLS server.  An unresponsive
       TLS server is dropped after approximately max-wait
       * max-attempts seconds.  The TLS client MUST send
       HeartbeatRequest messages, as defined by RFC 6520.";
    reference
      "RFC 6520: Transport Layer Security (TLS) and Datagram
       Transport Layer Security (DTLS) Heartbeat Extension";
    leaf max-wait {
      type uint16 {
        range "1..max";
      }
      units "seconds";
      default "30";
      description
        "Sets the amount of time in seconds after which if
         no data has been received from the TLS server, a
         TLS-level message will be sent to test the
         aliveness of the TLS server.";
    }
    leaf max-attempts {
      type uint8;
      default "3";
      description
        "Sets the maximum number of sequential keep-alive
         messages that can fail to obtain a response from
         the TLS server before assuming the TLS server is
```

```
              no longer alive.";
          }
        }
      }
    } // grouping tls-client-grouping
} // module ietf-tls-client
```

<CODE ENDS>

4. The "ietf-tls-server" Module

This section defines a YANG 1.1 [RFC7950] module called "ietf-tls-server". A high-level overview of the module is provided in Section 4.1. Examples illustatrating the module's use are provided in Examples (Section 4.2). The YANG module itself is defined in Section 4.3.

4.1. Data Model Overview

This section provides an overview of the "ietf-tls-server" module in terms of its features and groupings.

4.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-tls-server" module:

```
Features:
  +-- tls-server-hello-params-config
  +-- tls-server-keepalives
  +-- client-auth-config-supported
  +-- x509-certificate-auth
  +-- raw-public-key-auth
  +-- psk-auth
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

4.1.2. Groupings

The following diagram lists all the "grouping" statements defined in the "ietf-tls-server" module:

```
Groupings:
  +-- tls-server-grouping
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Each of these groupings are presented in the following subsections.

4.1.2.1. The "tls-server-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "tls-server-grouping" grouping:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

  grouping tls-server-grouping
    +-- server-identity
    |  +-- (auth-type)
    |     +--:(certificate) {x509-certificate-auth}?
    |     |  +-- certificate
    |     |     +---u ks:local-or-keystore-end-entity-cert-with-key-\
grouping
    |     +--:(raw-private-key) {raw-public-key-auth}?
    |     |  +-- raw-private-key
    |     |     +---u ks:local-or-keystore-asymmetric-key-grouping
    |     +--:(psk) {psk-auth}?
    |        +-- psk
    |           +---u ks:local-or-keystore-symmetric-key-grouping
    |           +-- id_hint?
    |                   string
    +-- client-authentication! {client-auth-config-supported}?
    |  +-- ca-certs! {x509-certificate-auth}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- ee-certs! {x509-certificate-auth}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- raw-public-keys! {raw-public-key-auth}?
    |  |  +---u ts:local-or-truststore-public-keys-grouping
    |  +-- psks?              empty {psk-auth}?
    +-- hello-params {tls-server-hello-params-config}?
    |  +---u tlscmn:hello-params-grouping
    +-- keepalives {tls-server-keepalives}?
       +-- peer-allowed-to-send?   empty
       +-- test-peer-aliveness!
          +-- max-wait?       uint16
          +-- max-attempts?   uint8
```

Comments:

*The "server-identity" node configures identity credentials, each of which is enabled by a "feature".

*The "client-authentication" node, which is optionally configured (as client authentication MAY occur at a higher protocol layer), configures trust anchors for authenticating the TLS client, with each option enabled by a "feature" statement.

*The "hello-params" node, which must be enabled by a feature, configures parameters for the TLS sessions established by this configuration.

*The "keepalives" node, which must be enabled by a feature, configures a flag enabling the TLS client to test the aliveness

of the TLS server, as well as a "presence" container for testing the aliveness of the TLSi client. The aliveness-tests occurs at the TLS protocol layer.

*For the referenced grouping statement(s):

- The "local-or-keystore-end-entity-cert-with-key-grouping" grouping is discussed in Section 2.1.3.6 of [I-D.ietf-netconf-keystore].
- The "local-or-keystore-asymmetric-key-grouping" grouping is discussed in Section 2.1.3.4 of [I-D.ietf-netconf-keystore].
- The "local-or-keystore-symmetric-key-grouping" grouping is discussed in Section 2.1.3.3 of [I-D.ietf-netconf-keystore].
- The "local-or-truststore-public-keys-grouping" grouping is discussed in Section 2.1.3.2 of [I-D.ietf-netconf-trust-anchors].
- The "local-or-truststore-certs-grouping" grouping is discussed in Section 2.1.3.1 of [I-D.ietf-netconf-trust-anchors].
- The "hello-params-grouping" grouping is discussed in Section 2.1.3.1 in this document.

4.1.3. Protocol-accessible Nodes

The "ietf-tls-server" module does not contain any protocol-accessible nodes.

4.2. Example Usage

This section presents two examples showing the "tls-server-grouping" grouping populated with some data. These examples are effectively the same except the first configures the server identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 2 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following configuration example uses local-definitions for the server identity and client authentication:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<tls-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server"
  xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">

  <!-- how this server will authenticate itself to the client -->
  <server-identity>
    <certificate>
      <local-definition>
        <public-key-format>ct:subject-public-key-info-format</public\
-key-format>
        <public-key>base64encodedvalue==</public-key>
        <private-key-format>ct:rsa-private-key-format</private-key-f\
ormat>
        <cleartext-private-key>base64encodedvalue==</cleartext-priva\
te-key>
        <cert-data>base64encodedvalue==</cert-data>
      </local-definition>
    </certificate>
    <!-- TESTED, BUT COMMENTED OUT DUE TO ONLY ONE ALLOWED AT A TIME
    <raw-private-key>
      <local-definition>
        <public-key-format>ct:subject-public-key-info-format</public\
-key-format>
        <public-key>base64encodedvalue==</public-key>
        <private-key-format>ct:rsa-private-key-format</private-key-f\
ormat>
        <cleartext-private-key>base64encodedvalue==</cleartext-priva\
te-key>
      </local-definition>
    </raw-private-key>
    <psk>
      <local-definition>
        <key-format>ct:octet-string-key-format</key-format>
        <cleartext-key>base64encodedvalue==</cleartext-key>
      </local-definition>
    </psk>
    -->
  </server-identity>

  <!-- which certificates will this server trust -->
  <client-authentication>
    <ca-certs>
      <local-definition>
        <certificate>
          <name>Identity Cert Issuer #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
```

```
        <certificate>
          <name>Identity Cert Issuer #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ca-certs>
    <ee-certs>
      <local-definition>
        <certificate>
          <name>Application #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>Application #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ee-certs>
    <raw-public-keys>
      <local-definition>
        <public-key>
          <name>User A</name>
          <public-key-format>ct:subject-public-key-info-format</publ\
ic-key-format>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
        <public-key>
          <name>User B</name>
          <public-key-format>ct:subject-public-key-info-format</publ\
ic-key-format>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
      </local-definition>
    </raw-public-keys>
    <psks/>
  </client-authentication>

  <keepalives>
    <peer-allowed-to-send/>
  </keepalives>

</tls-server>
```

The following configuration example uses keystore-references for the server identity and truststore-references for client authentication: from the keystore:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<tls-server xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">

  <!-- how this server will authenticate itself to the client -->
  <server-identity>
    <certificate>
      <keystore-reference>
        <asymmetric-key>rsa-asymmetric-key</asymmetric-key>
        <certificate>ex-rsa-cert</certificate>
      </keystore-reference>
    </certificate>
    <!-- TESTED, BUT COMMENTED OUT DUE TO ONLY ONE ALLOWED AT A TIME
    <raw-private-key>
      <keystore-reference>raw-private-key</keystore-reference>
    </raw-private-key>
    <psk>
      <keystore-reference>encrypted-symmetric-key</keystore-referenc\
e>
    </psk>
    -->
  </server-identity>

  <!-- which certificates will this server trust -->
  <client-authentication>
    <ca-certs>
      <truststore-reference>trusted-client-ca-certs</truststore-refe\
rence>
    </ca-certs>
    <ee-certs>
      <truststore-reference>trusted-client-ee-certs</truststore-refe\
rence>
    </ee-certs>
    <raw-public-keys>
      <truststore-reference>Raw Public Keys for TLS Clients</truststo\
ore-reference>
    </raw-public-keys>
    <psks/>
  </client-authentication>

  <keepalives>
    <peer-allowed-to-send/>
  </keepalives>

</tls-server>
```

4.3. YANG Module

This YANG module has a normative references to [RFC5246], [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-tls-server@2020-08-20.yang"

```
module ietf-tls-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
  prefix tlss;

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC AAAA: YANG Data Types and Groupings for Cryptography";
  }

  import ietf-truststore {
    prefix ts;
    reference
      "RFC BBBB: A YANG Data Model for a Truststore";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC CCCC: A YANG Data Model for a Keystore";
  }

  import ietf-tls-common {
    prefix tlscmn;
    revision-date 2020-08-20; // stable grouping definitions
    reference
      "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kent+ietf@watsen.net>
     Author:   Gary Wu <mailto:garywu@cisco.com>";

  description
    "This module defines reusable groupings for TLS servers that
     can be used as a basis for specific TLS server instances.

     Copyright (c) 2020 IETF Trust and the persons identified
```

```
   as authors of the code. All rights reserved.

   Redistribution and use in source and binary forms, with
   or without modification, is permitted pursuant to, and
   subject to the license terms contained in, the Simplified
   BSD License set forth in Section 4.c of the IETF Trust's
   Legal Provisions Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC FFFF
   (https://www.rfc-editor.org/info/rfcFFFF); see the RFC
   itself for full legal notices.

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
   'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
   'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
   are to be interpreted as described in BCP 14 (RFC 2119)
   (RFC 8174) when, and only when, they appear in all
   capitals, as shown here.";

 revision 2020-08-20 {
   description
     "Initial version";
   reference
     "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
 }

 // Features

 feature tls-server-hello-params-config {
   description
     "TLS hello message parameters are configurable on a TLS
      server.";
 }

 feature tls-server-keepalives {
   description
     "Per socket TLS keepalive parameters are configurable for
      TLS servers on the server implementing this feature.";
 }

 feature client-auth-config-supported {
   description
     "Indicates that the configuration for how to authenticate
      clients can be configured herein, as opposed to in an
      application specific location.  That is, to support the
      consuming data models that prefer to place client
      authentication with client definitions, rather then
      in a data model principally concerned with configuring
      the transport.";
```

```
}

feature x509-certificate-auth {
  description
    "Indicates that the server supports authenticating clients
     using X.509 certificates.";
}

feature raw-public-key-auth {
  description
    "Indicates that the server supports authenticating clients
     using ray public keys.";
}

feature psk-auth {
  description
    "Indicates that the server supports authenticating clients
     using PSKs (pre-shared or pairwise-symmetric keys).";
}



// Groupings

grouping tls-server-grouping {
  description
    "A reusable grouping for configuring a TLS server without
     any consideration for how underlying TCP sessions are
     established.

     Note that this grouping uses fairly typical descendent
     node names such that a stack of 'uses' statements will
     have name conflicts.  It is intended that the consuming
     data model will resolve the issue (e.g., by wrapping
     the 'uses' statement in a container called
     'tls-server-parameters').  This model purposely does
     not do this itself so as to provide maximum flexibility
     to consuming models.";


  container server-identity {
    nacm:default-deny-write;
    description
      "A locally-defined or referenced end-entity certificate,
       including any configured intermediate certificates, the
       TLS server will present when establishing a TLS connection
       in its Certificate message, as defined in Section 7.4.2
       in RFC 5246.";
    reference
      "RFC 5246: The Transport Layer Security (TLS) Protocol
```

```
            Version 1.2
   RFC CCCC: A YANG Data Model for a Keystore";
choice auth-type {
  mandatory true;
  description
    "A choice amongst authentication types.";
  case certificate {
    if-feature x509-certificate-auth;
    container certificate {
      description
        "Specifies the server identity using a certificate.";
      uses
        ks:local-or-keystore-end-entity-cert-with-key-grouping{
        refine "local-or-keystore/local/local-definition" {
          must 'public-key-format'
               + ' = "ct:subject-public-key-info-format"';
        }
        refine "local-or-keystore/keystore/keystore-reference"
               + "/asymmetric-key" {
          must 'deref(.)/../ks:public-key-format'
               + ' = "ct:subject-public-key-info-format"';
        }
      }
    }
  }
  case raw-private-key {
    if-feature raw-public-key-auth;
    container raw-private-key {
      description
        "Specifies the server identity using a raw
         private key.";
      uses ks:local-or-keystore-asymmetric-key-grouping {
        refine "local-or-keystore/local/local-definition" {
          must 'public-key-format'
               + ' = "ct:subject-public-key-info-format"';
        }
        refine "local-or-keystore/keystore/keystore-reference"{
          must 'deref(.)/../ks:public-key-format'
               + ' = "ct:subject-public-key-info-format"';
        }
      }
    }
  }
  case psk {
    if-feature psk-auth;
    container psk {
      description
        "Specifies the server identity using a PSK (pre-shared
         or pairwise-symmetric key).";
```

```
        uses ks:local-or-keystore-symmetric-key-grouping;
        leaf id_hint {
          type string;
          description
            "The key 'psk_identity_hint' value used in the TLS
             'ServerKeyExchange' message.";
          reference
            "RFC 4279: Pre-Shared Key Ciphersuites for
                       Transport Layer Security (TLS)";
        }
      }
    }
  }
} // container server-identity

container client-authentication {
  if-feature "client-auth-config-supported";
  nacm:default-deny-write;
  must 'ca-certs or ee-certs or raw-public-keys or psks';
  presence
    "Indicates that client authentication is supported (i.e.,
     that the server will request clients send certificates).
     If not configured, the TLS server SHOULD NOT request the
     TLS clients provide authentication credentials.";
  description
    "Specifies how the TLS server can authenticate TLS clients.
     Any combination of credentials is additive and unordered.

     Note that no configuration is required for PSK (pre-shared
     or pairwise-symmetric key) based authentication as the key
     is necessarily the same as configured in the '../server-
     identity' node.";
  container ca-certs {
    if-feature "x509-certificate-auth";
    presence
      "Indicates that the TLS server can authenticate TLS clients
       using configured certificate authority certificates.";
    description
      "A set of certificate authority (CA) certificates used by
       the TLS server to authenticate TLS client certificates. A
       client certificate is authenticated if it has a valid
       chain of trust to a configured CA certificate.";
    reference
      "RFC BBBB: A YANG Data Model for a Truststore";
    uses ts:local-or-truststore-certs-grouping;
  }
  container ee-certs {
    if-feature "x509-certificate-auth";
    presence
```

```
    "Indicates that the TLS server can authenticate TLS
     clients using configured client certificates.";
  description
    "A set of client certificates (i.e., end entity
     certificates) used by the TLS server to authenticate
     certificates presented by TLS clients. A client
     certificate is authenticated if it is an exact
     match to a configured client certificate.";
  reference
    "RFC BBBB: A YANG Data Model for a Truststore";
  uses ts:local-or-truststore-certs-grouping;
}
container raw-public-keys {
  if-feature "raw-public-key-auth";
  presence
    "Indicates that the TLS server can authenticate TLS
     clients using raw public keys.";
  description
    "A set of raw public keys used by the TLS server to
     authenticate raw public keys presented by the TLS
     client.  A raw public key is authenticated if it
     is an exact match to a configured raw public key.";
  reference
    "RFC BBBB: A YANG Data Model for a Truststore";
  uses ts:local-or-truststore-public-keys-grouping {
    refine "local-or-truststore/local/local-definition"
           + "/public-key" {
      must 'public-key-format'
           + ' = "ct:subject-public-key-info-format"';
    }
    refine "local-or-truststore/truststore"
           + "/truststore-reference" {
      must 'deref(.)/../*/ts:public-key-format'
           + ' = "ct:subject-public-key-info-format"';
    }
  }
}
leaf psks {
  if-feature "psk-auth";
  type empty;
  description
    "Indicates that the TLS server can authenticate TLS clients
     using configured PSKs (pre-shared or pairwise-symmetric
     keys).

     No configuration is required since the PSK value is the
     same as PSK value configured in the 'server-identity'
     node.";
}
```

```
} // container client-authentication

container hello-params {
  nacm:default-deny-write;
  if-feature "tls-server-hello-params-config";
  uses tlscmn:hello-params-grouping;
  description
    "Configurable parameters for the TLS hello message.";
} // container hello-params

container keepalives {
  nacm:default-deny-write;
  if-feature "tls-server-keepalives";
  description
    "Configures the keepalive policy for the TLS server.";
  leaf peer-allowed-to-send {
    type empty;
    description
      "Indicates that the remote TLS client is allowed to send
       HeartbeatRequest messages, as defined by RFC 6520
       to this TLS server.";
    reference
      "RFC 6520: Transport Layer Security (TLS) and Datagram
       Transport Layer Security (DTLS) Heartbeat Extension";
  }
  container test-peer-aliveness {
    presence
      "Indicates that the TLS server proactively tests the
       aliveness of the remote TLS client.";
    description
      "Configures the keep-alive policy to proactively test
       the aliveness of the TLS client.  An unresponsive
       TLS client is dropped after approximately max-wait
       * max-attempts seconds.";
    leaf max-wait {
      type uint16 {
        range "1..max";
      }
      units "seconds";
      default "30";
      description
        "Sets the amount of time in seconds after which if
         no data has been received from the TLS client, a
         TLS-level message will be sent to test the
         aliveness of the TLS client.";
    }
    leaf max-attempts {
      type uint8;
      default "3";
```

```
          description
            "Sets the maximum number of sequential keep-alive
             messages that can fail to obtain a response from
             the TLS client before assuming the TLS client is
             no longer alive.";
        }
      }
    } // container keepalives
  } // grouping tls-server-grouping
} // module ietf-tls-server
```

<CODE ENDS>

5. Security Considerations

5.1. The "ietf-tls-common" YANG Module

The "ietf-tls-common" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

None of the writable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-write" extension has not been set for any data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

5.2. The "ietf-tls-client" YANG Module

The "ietf-tls-client" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

Please be aware that this module uses the "key" and "private-key" nodes from the "ietf-crypto-types" module [I-D.ietf-netconf-crypto-types], where said nodes have the NACM extension "default-deny-all" set, thus preventing unrestricted read-access to the cleartext key values.

All of the writable data nodes defined by this module may be considered sensitive or vulnerable in some network environments. For instance, any modification to a key or reference to a key may dramatically alter the implemented security policy. For this reason, the NACM extension "default-deny-write" has been set for all data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

5.3. The "ietf-tls-server" YANG Module

The "ietf-tls-server" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

Please be aware that this module uses the "key" and "private-key" nodes from the "ietf-crypto-types" module [I-D.ietf-netconf-crypto-types], where said nodes have the NACM extension "default-deny-all" set, thus preventing unrestricted read-access to the cleartext key values.

All of the writable data nodes defined by this module may be considered sensitive or vulnerable in some network environments. For

instance, any modification to a key or reference to a key may dramatically alter the implemented security policy. For this reason, the NACM extension "default-deny-write" has been set for all data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

6. IANA Considerations

6.1. The "IETF XML" Registry

This document registers three URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tls-common
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tls-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tls-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

6.2. The "YANG Module Names" Registry

This document registers three YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

```
name:         ietf-tls-common
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-common
prefix:       tlscmn
reference:    RFC FFFF

name:         ietf-tls-client
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-client
prefix:       tlsc
reference:    RFC FFFF

name:         ietf-tls-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-server
prefix:       tlss
reference:    RFC FFFF
```

7. References

7.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "YANG Data Types and Groupings for Cryptography", Work in Progress, Internet-Draft, draft-ietf-netconf-crypto-types-17, 10 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-crypto-types-17>.

[I-D.ietf-netconf-keystore] Watsen, K., "A YANG Data Model for a Keystore", Work in Progress, Internet-Draft, draft-ietf-netconf-keystore-19, 10 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-keystore-19>.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "A YANG Data Model for a Truststore", Work in Progress, Internet-Draft, draft-ietf-netconf-trust-anchors-12, 10 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-trust-anchors-12>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <https://www.rfc-editor.org/info/rfc5288>.

[RFC5289] Rescorla, E., "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", RFC 5289, DOI 10.17487/RFC5289, August 2008, <https://www.rfc-editor.org/info/rfc5289>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <https://www.rfc-editor.org/info/rfc7589>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

7.2. Informative References

[I-D.ietf-netconf-http-client-server]
Watsen, K., "YANG Groupings for HTTP Clients and HTTP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-http-client-server-04, 8 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-http-client-server-04>.

[I-D.ietf-netconf-netconf-client-server]
Watsen, K., "NETCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-netconf-client-server-20, 8 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-netconf-client-server-20>.

[I-D.ietf-netconf-restconf-client-server]
Watsen, K., "RESTCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-client-server-20, 8 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-restconf-client-server-20>.

[I-D.ietf-netconf-ssh-client-server]
Watsen, K. and G. Wu, "YANG Groupings for SSH Clients and SSH Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-ssh-client-server-21, 10 July 2020,

<https://tools.ietf.org/html/draft-ietf-netconf-ssh-client-server-21>.

[I-D.ietf-netconf-tcp-client-server]
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-07, 8 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-tcp-client-server-07>.

[I-D.ietf-netconf-tls-client-server]
Watsen, K. and G. Wu, "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-21, 10 July 2020, <https://tools.ietf.org/html/draft-ietf-netconf-tls-client-server-21>.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, DOI 10.17487/RFC2246, January 1999, <https://www.rfc-editor.org/info/rfc2246>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, DOI 10.17487/RFC4346, April 2006, <https://www.rfc-editor.org/info/rfc4346>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol

(NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <https://www.rfc-editor.org/info/rfc8071>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

Appendix A. Change Log

This section is to be removed before publishing as an RFC.

A.1. 00 to 01

* Noted that '0.0.0.0' and '::' might have special meanings.

* Renamed "keychain" to "keystore".

A.2. 01 to 02

* Removed the groupings containing transport-level configuration. Now modules contain only the transport-independent groupings.

* Filled in previously incomplete 'ietf-tls-client' module.

* Added cipher suites for various algorithms into new 'ietf-tls-common' module.

A.3. 02 to 03

* Added a 'must' statement to container 'server-auth' asserting that at least one of the various auth mechanisms must be specified.

* Fixed description statement for leaf 'trusted-ca-certs'.

A.4. 03 to 04

*Updated title to "YANG Groupings for TLS Clients and TLS Servers"

*Updated leafref paths to point to new keystore path

*Changed the YANG prefix for ietf-tls-common from 'tlscom' to 'tlscmn'.

*Added TLS protocol verions 1.0 and 1.1.

*Made author lists consistent

*Now tree diagrams reference ietf-netmod-yang-tree-diagrams

*Updated YANG to use typedefs around leafrefs to common keystore paths

*Now inlines key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

*Merged changes from co-author.

A.6. 05 to 06

*Updated to use trust anchors from trust-anchors draft (was keystore draft)

*Now Uses new keystore grouping enabling asymmetric key to be either locally defined or a reference to the keystore.

A.7. 06 to 07

*factored the tls-[client|server]-groupings into more reusable groupings.

*added if-feature statements for the new "x509-certificates" feature defined in draft-ietf-netconf-trust-anchors.

A.8. 07 to 08

*Added a number of compatibility matrices to Section 5 (thanks Frank!)

*Clarified that any configured "cipher-suite" values need to be compatible with the configured private key.

A.9. 08 to 09

* Updated examples to reflect update to groupings defined in the keystore draft.

* Add TLS keepalives features and groupings.

* Prefixed top-level TLS grouping nodes with 'tls-' and support mashups.

* Updated copyright date, boilerplate template, affiliation, and folding algorithm.

A.10. 09 to 10

* Reformatted the YANG modules.

A.11. 10 to 11

* Collapsed all the inner groupings into the top-level grouping.

* Added a top-level "demux container" inside the top-level grouping.

* Added NACM statements and updated the Security Considerations section.

* Added "presence" statements on the "keepalive" containers, as was needed to address a validation error that appeared after adding the "must" statements into the NETCONF/RESTCONF client/server modules.

* Updated the boilerplate text in module-level "description" statement to match copyeditor convention.

A.12. 11 to 12

* In server model, made 'client-authentication' a 'presence' node indicating that the server supports client authentication.

* In the server model, added a 'required-or-optional' choice to 'client-authentication' to better support protocols such as RESTCONF.

* In the server model, added a 'local-or-external' choice to 'client-authentication' to better support consuming data models that prefer to keep client auth with client definitions than in a model principally concerned with the "transport".

*In both models, removed the "demux containers", floating the nacm:default-deny-write to each descendent node, and adding a note to model designers regarding the potential need to add their own demux containers.

*Fixed a couple references (section 2 --> section 3)

A.13. 12 to 13

*Updated to reflect changes in trust-anchors drafts (e.g., s/trust-anchors/truststore/g + s/pinned.//)

A.14. 12 to 13

*Removed 'container' under 'client-identity' to match server model.

*Updated examples to reflect change grouping in keystore module.

A.15. 13 to 14

*Removed the "certificate" container from "client-identity" in the ietf-tls-client module.

*Updated examples to reflect ietf-crypto-types change (e.g., identities --> enumerations)

A.16. 14 to 15

*Updated "server-authentication" and "client-authentication" nodes from being a leaf of type "ts:certificates-ref" to a container that uses "ts:local-or-truststore-certs-grouping".

A.17. 15 to 16

*Removed unnecessary if-feature statements in the -client and -server modules.

*Cleaned up some description statements in the -client and -server modules.

*Fixed a canonical ordering issue in ietf-tls-common detected by new pyang.

A.18. 16 to 17

*Removed choice local-or-external by removing the 'external' case and flattening the 'local' case and adding a "client-auth-config-supported" feature.

*Removed choice required-or-optional.

Updated examples to include the "-key-format" nodes.

*Augmented-in "must" expressions ensuring that locally-defined public-key-format are "ct:ssh-public-key-format" (must expr for ref'ed keys are TBD).

A.19. 17 to 18

*Removed the unused "external-client-auth-supported" feature.

*Made client-indentity optional, as there may be over-the-top auth instead.

*Added augment to uses of local-or-keystore-symmetric-key-grouping for a psk "id" node.

*Added missing presence container "psks" to ietf-tls-server's "client-authentication" container.

*Updated examples to reflect new "bag" addition to truststore.

*Removed feature-limited caseless 'case' statements to improve tree diagram rendering.

*Refined truststore/keystore groupings to ensure the key formats "must" be particular values.

*Switched to using truststore's new "public-key" bag (instead of separate "ssh-public-key" and "raw-public-key" bags.

*Updated client/server examples to cover ALL cases (local/ref x cert/raw-key/psk).

A.20. 18 to 19

*Updated the "keepalives" containers in part to address Michal Vasko's request to align with RFC 8071, and in part to better align to RFC 6520.

*Removed algorithm-mapping tables from the "TLS Common Model" section

*Removed the 'algorithm' node from the examples.

*Renamed both "client-certs" and "server-certs" to "ee-certs"

*Added a "Note to Reviewers" note to first page.

A.21. 19 to 20

* Modified the 'must' expression in the "ietf-tls-client:server-authention" node to cover the "raw-public-keys" and "psks" nodes also.

* Added a "must 'ca-certs or ee-certs or raw-public-keys or psks'" statement to the ietf-tls-server:client-authentication" node.

* Added "mandatory true" to "choice auth-type" and a "presence" statement to its ancestor.

* Expanded "Data Model Overview section(s) [remove "wall" of tree diagrams].

* Moved the "ietf-ssh-common" module section to proceed the other two module sections.

* Updated the Security Considerations section.

A.22. 20 to 21

* Updated examples to reflect new "cleartext-" prefix in the crypto-types draft.

A.23. 21 to 22

* In both the "client-authentication" and "server-authentication" subtrees, replaced the "psks" node from being a P-container to a leaf of type "empty".

* Cleaned up examples (e.g., removed FIXMEs)

* Fixed issues found by the SecDir review of the "keystore" draft.

* Updated the "psk" sections in the "ietf-tls-client" and "ietf-tls-server" modules to more correctly reflect RFC 4279.

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by first name): Alan Luchuk, Andy Bierman, Balazs Kovacs, Benoit Claise, Bert Wijnen, David Lamparter, Gary Wu, Henk Birkholz, Juergen Schoenwaelder, Ladislav Lhotka, Liang Xia, Martin Bjorklund, Mehmet Ersue, Michal Vasko, Phil Shafer, Radek Krejci, Sean Turner, and Tom Petch.

Special acknowledgement goes to Gary Wu who contributed the "ietf-tls-common" module.

Author's Address

Kent Watsen
Watsen Networks

Email: kent+ietf@watsen.net