NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2020

K. Watsen
Watsen Networks
G. Wu
Cisco Systems
L. Xia
Huawei

March 8, 2020

YANG Groupings for TLS Clients and TLS Servers
draft-ietf-netconf-tls-client-server-18

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 many 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.

This document contains references to other drafts in progress, both in the Normative References section, as well as in body text throughout. Please update the following references to reflect their final RFC assignments:

- I-D.ietf-netconf-trust-anchors

- I-D.ietf-netconf-keystore

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

- "XXXX" --> the assigned RFC value for this draft

- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-trust-anchors

- "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-keystore

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

o "2020-03-08" --> the publication date of this draft

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

o 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 September 9, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Terminology 4
3. The TLS Client Model 4
  3.1. Tree Diagram 4
  3.2. Example Usage 5
  3.3. YANG Module 9
4. The TLS Server Model 16
  4.1. Tree Diagram 16

4.2. Example Usage 17
4.3. YANG Module 19
5. The TLS Common Model 27
5.1. Tree Diagram 36
5.2. Example Usage 36
5.3. YANG Module 36
6. Security Considerations 45
7. IANA Considerations 46
7.1. The IETF XML Registry 46
7.2. The YANG Module Names Registry 47
8. References 47
8.1. Normative References 47
8.2. Informative References 49
Appendix A. Change Log 51
A.1. 00 to 01 51
A.2. 01 to 02 51
A.3. 02 to 03 51
A.4. 03 to 04 51
A.5. 04 to 05 52
A.6. 05 to 06 52
A.7. 06 to 07 52
A.8. 07 to 08 52
A.9. 08 to 09 52
A.10. 09 to 10 52
A.11. 10 to 11 53
A.12. 11 to 12 53
A.13. 12 to 13 53
A.14. 12 to 13 53
A.15. 13 to 14 54
A.16. 14 to 15 54
A.17. 15 to 16 54
A.18. 16 to 17 54
A.19. 17 to 18 54
Acknowledgements 55
Authors' Addresses 55

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.

2. Terminology

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.

3. The TLS Client Model

3.1. Tree Diagram

This section provides a tree diagram [RFC8340] for the "ietf-tls-client" module that does not have groupings expanded.

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

module: ietf-tls-client

  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
    +-- server-authentication
    |  +-- ca-certs! {x509-certificate-auth}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- server-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! {psk-auth}?
    +-- hello-params {tls-client-hello-params-config}?
    |  +---u tlscmn:hello-params-grouping
    +-- keepalives! {tls-client-keepalives}?
       +-- max-wait?       uint16
       +-- max-attempts?   uint8
```

3.2. Example Usage

This section presents two examples showing the tls-client-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 example configures the client identity using a local key:

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

<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>
           <algorithm>rsa2048</algorithm>
           <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>
           <private-key>base64encodedvalue==</private-key>
           <cert>base64encodedvalue==</cert>
         </local-definition>
       </certificate>
       <!-- TESTED, BUT COMMENTED OUT DUE TO ONLY ONE ALLOWED AT A TIME
       <raw-private-key>
         <local-definition>
           <algorithm>rsa2048</algorithm>
           <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>
           <private-key>base64encodedvalue==</private-key>
         </local-definition>
       </raw-private-key>
       <psk>
         <local-definition>
           <algorithm>aes-256-cbc</algorithm>
           <key-format>ct:octet-string-key-format</key-format>
           <key>base64encodedvalue==</key>
         </local-definition>
       </psk>
       -->
     </client-identity>

     <!-- which certificates will this client trust -->
     <server-authentication>
       <ca-certs>
         <local-definition>
           <cert>base64encodedvalue==</cert>
           <cert>base64encodedvalue==</cert>
           <cert>base64encodedvalue==</cert>
         </local-definition>
       </ca-certs>
       <server-certs>
```

```
        <local-definition>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
        </local-definition>
      </server-certs>
      <raw-public-keys>
        <local-definition>
          <public-key>
            <name>corp-fw1</name>
            <algorithm>secp256r1</algorithm>
            <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>
            <algorithm>secp256r1</algorithm>
            <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>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </keepalives>

  </tls-client>
```

The following example configures the client identity using a key from the keystore:

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

<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>
    <server-certs>
      <truststore-reference>trusted-server-ee-certs</truststore-refe\
rence>
    </server-certs>
    <raw-public-keys>
      <truststore-reference>Raw Public Keys for Servers</truststore-\
reference>
    </raw-public-keys>
    <psks/>
  </server-authentication>

  <keepalives>
    <max-wait>30</max-wait>
    <max-attempts>3</max-attempts>
  </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@2020-03-08.yang"

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

  import ietf-tls-common {
    prefix tlscmn;
    revision-date 2020-03-08; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC AAAA: Common YANG Data Types 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-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  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) 2019 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 XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); 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 2020-03-08 {
       description
         "Initial version";
       reference
         "RFC XXXX: 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;
         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 ZZZZ:
              YANG Data Model for a 'Keystore' Mechanism";
         choice auth-type {
           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).  Note that, when the PSK is
                configured as a Keystore reference, the key's 'name'
                node MAY be used as the PSK's ID when used by the TLS
                protocol.";
              uses ks:local-or-keystore-symmetric-key-grouping {
                augment "local-or-keystore/local/local-definition" {
                  if-feature "ks:local-definitions-supported";
                  description
                    "Adds an 'id' value when the PSK is used by TLS.";
                  leaf id {
                    type string;  // is this the right type?
                    description
                      "The key id used in the TLS protocol for PSKs.";
                  }
                }
              }
            }
          }
        }
      } // container client-identity

      container server-authentication {
        nacm:default-deny-write;
        must 'ca-certs or server-certs';
        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 YYYY: YANG Data Model for a Truststore";
          uses ts:local-or-truststore-certs-grouping;
            // Note: TS certs don't have a key-format...no test needed
        }
        container server-certs {
          // FIXME: plural too much?  rename to 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 YYYY: YANG Data Model for a Truststore";
          uses ts:local-or-truststore-certs-grouping;
            // Note: TS certs don't have a key-format...no test needed
        }
        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 YYYY: 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"';
            }
          }
        }
        container psks {
          if-feature "psk-auth";
```

```
          presence
            "Indicates that the TLS client can authenticate TLS servers
             using a configure PSK (pre-shared or pairwise-symmetric
             key).";
          description
            "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";
        presence "Indicates that keepalives are enabled.";
        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.";
        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 TLS Server Model

4.1. Tree Diagram

This section provides a tree diagram [RFC8340] for the "ietf-tls-server" module that does not have groupings expanded.

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

module: ietf-tls-server

  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
    +-- client-authentication! {client-auth-config-supported}?
    |  +-- ca-certs! {x509-certificate-auth}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- client-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! {psk-auth}?
    +-- hello-params {tls-server-hello-params-config}?
    |  +---u tlscmn:hello-params-grouping
    +-- keepalives! {tls-server-keepalives}?
       +-- max-wait?       uint16
       +-- max-attempts?   uint8
```

4.2. Example Usage

This section presents two examples showing the tls-server-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 example configures the server identity using a local key:

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

<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>
        <algorithm>rsa2048</algorithm>
        <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>
        <private-key>base64encodedvalue==</private-key>
        <cert>base64encodedvalue==</cert>
      </local-definition>
    </certificate>
    <!-- TESTED, BUT COMMENTED OUT DUE TO ONLY ONE ALLOWED AT A TIME
    <raw-private-key>
      <local-definition>
        <algorithm>rsa2048</algorithm>
        <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>
        <private-key>base64encodedvalue==</private-key>
      </local-definition>
    </raw-private-key>
    <psk>
      <local-definition>
        <algorithm>aes-256-cbc</algorithm>
```

```
          <key-format>ct:octet-string-key-format</key-format>
          <key>base64encodedvalue==</key>
        </local-definition>
      </psk>
      -->
    </server-identity>

    <!-- which certificates will this server trust -->
    <client-authentication>
      <ca-certs>
        <local-definition>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
        </local-definition>
      </ca-certs>
      <client-certs>
        <local-definition>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
        </local-definition>
      </client-certs>
      <raw-public-keys>
        <local-definition>
          <public-key>
            <name>User A</name>
            <algorithm>secp256r1</algorithm>
            <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>
            <algorithm>secp256r1</algorithm>
            <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>

  </tls-server>
```

The following example configures the server identity using a key from the keystore:

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

<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>
    <client-certs>
      <truststore-reference>trusted-client-ee-certs</truststore-refe\
rence>
    </client-certs>
    <raw-public-keys>
      <truststore-reference>Raw Public Keys for Clients</truststore-\
reference>
    </raw-public-keys>
    <psks/>
  </client-authentication>

</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@2020-03-08.yang"
```

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

  import ietf-tls-common {
    prefix tlscmn;
    revision-date 2020-03-08; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC AAAA: Common YANG Data Types 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-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  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) 2019 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 XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); 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 2020-03-08 {
       description
         "Initial version";
       reference
         "RFC XXXX: 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 ZZZZ:
              YANG Data Model for a 'Keystore' Mechanism";
         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).  Note that, when the PSK is
                configured as a Keystore reference, the key's 'name'
                node MAY be used as the PSK's ID when used by the TLS
                protocol.";
              uses ks:local-or-keystore-symmetric-key-grouping {
                augment "local-or-keystore/local/local-definition" {
                  if-feature "ks:local-definitions-supported";
                  description
                    "An 'id' value for when the PSK is used by TLS.";
                  leaf id {
                    type string;  // is this the right type?
                    description
                      "The key id used in the TLS protocol for PSKs.";
                  }
                }
              }
            }
          }
        }
      } // container server-identity

      container client-authentication {
        if-feature "client-auth-config-supported";
        nacm:default-deny-write;
        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 CCCC: YANG Data Model for a Truststore";
            uses ts:local-or-truststore-certs-grouping;
              // Note: TS certs don't have a key-format...no test needed
          }
          container client-certs {   // FIXME: plural too much?
            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 CCCC: YANG Data Model for a Truststore";
            uses ts:local-or-truststore-certs-grouping;
              // Note: TS certs don't have a key-format...no test needed
          }
          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 CCCC: 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"';
```

```
            }
          }
        }
        container psks {
          if-feature "psk-auth";
          presence
            "Indicates that the TLS server can authenticate the TLS
             client using its PSK (pre-shared or pairwise-symmetric
             key).";
          description
            "No configuration is required since the PSK value is the
             same as PSK value configured in the 'client-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";
        presence "Indicates that keepalives are enabled.";
        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. The TLS Common Model

The TLS common model presented in this section contains identities and groupings common to both TLS clients and TLS servers. The hello-params-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):

- o 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;

- o 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 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.

[I-D.ietf-netconf-crypto-types] defines six categories of cryptographic algorithms (hash-algorithm, symmetric-key-encryption-algorithm, mac-algorithm, asymmetric-key-encryption-algorithm, signature-algorithm, key-negotiation-algorithm) and lists several widely accepted algorithms for each of them. The TLS client and server models use one or more of these algorithms. The following tables are provided, in part to define the subset of algorithms defined in the crypto-types model used by TLS, and in part to ensure compatibility of configured TLS cryptographic parameters for configuring its permitted TLS algorithms:

ciper-suites in hello-params-grouping	HASH
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	sha-256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	sha-384
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	sha-256
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	sha-384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	sha-256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	sha-384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	sha-256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	sha-384
TLS_DHE_RSA_WITH_AES_128_CCM	sha-256
TLS_DHE_RSA_WITH_AES_256_CCM	sha-256
TLS_DHE_PSK_WITH_AES_128_CCM	sha-256
TLS_DHE_PSK_WITH_AES_256_CCM	sha-256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	sha-256
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	sha-384
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	sha-256

Table 1-1 TLS 1.2 Compatibility Matrix Part 1: ciper-suites mapping to hash-algorithm

ciper-suites in hello-params-grouping	symmetric
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	enc-aes-256-gcm
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	enc-aes-256-gcm
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	enc-aes-256-gcm
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	enc-aes-256-gcm
TLS_DHE_RSA_WITH_AES_128_CCM	enc-aes-128-ccm
TLS_DHE_RSA_WITH_AES_256_CCM	enc-aes-256-ccm
TLS_DHE_PSK_WITH_AES_128_CCM	enc-aes-128-ccm
TLS_DHE_PSK_WITH_AES_256_CCM	enc-aes-256-ccm
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	enc-aes-256-gcm
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	enc-aes-128-ccm

Table 1-2 TLS 1.2 Compatibility Matrix Part 2: ciper-suites mapping to symmetric-key-encryption-algorithm

ciper-suites in hello-params-grouping	MAC
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	mac-aes-128-gcm
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	mac-aes-256-gcm
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	mac-aes-128-gcm
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	mac-aes-256-gcm
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	mac-aes-128-gcm
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	mac-aes-256-gcm
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	mac-aes-128-gcm
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	mac-aes-256-gcm
TLS_DHE_RSA_WITH_AES_128_CCM	mac-aes-128-ccm
TLS_DHE_RSA_WITH_AES_256_CCM	mac-aes-256-ccm
TLS_DHE_PSK_WITH_AES_128_CCM	mac-aes-128-ccm
TLS_DHE_PSK_WITH_AES_256_CCM	mac-aes-256-ccm
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	mac-chacha20-poly1305
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	mac-chacha20-poly1305
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	mac-chacha20-poly1305
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	mac-chacha20-poly1305
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	mac-chacha20-poly1305
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	mac-aes-128-gcm
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	mac-aes-256-gcm
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	mac-aes-128-ccm

Table 1-3 TLS 1.2 Compatibility Matrix Part 3: ciper-suites mapping to MAC-algorithm

ciper-suites in hello-params-grouping	signature
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	rsa-pkcs1-sha256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	rsa-pkcs1-sha384
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	N/A
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	N/A
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ecdsa-secp256r1-sha256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ecdsa-secp384r1-sha384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	rsa-pkcs1-sha256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	rsa-pkcs1-sha384
TLS_DHE_RSA_WITH_AES_128_CCM	rsa-pkcs1-sha256
TLS_DHE_RSA_WITH_AES_256_CCM	rsa-pkcs1-sha256
TLS_DHE_PSK_WITH_AES_128_CCM	N/A
TLS_DHE_PSK_WITH_AES_256_CCM	N/A
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	rsa-pkcs1-sha256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	ecdsa-secp256r1-sha256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	rsa-pkcs1-sha256
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	N/A
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	N/A
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	N/A
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	N/A
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	N/A

Table 1-4 TLS 1.2 Compatibility Matrix Part 4: ciper-suites mapping to signature-algorithm

ciper-suites in hello-params-grouping	key-negotiation
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	dhe-ffdhe2048, ...
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	dhe-ffdhe2048, ...
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	psk-dhe-ffdhe2048, ...
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	psk-dhe-ffdhe2048, ...
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ecdhe-secp256r1, ...
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ecdhe-secp256r1, ...
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ecdhe-secp256r1, ...
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ecdhe-secp256r1, ...
TLS_DHE_RSA_WITH_AES_128_CCM	dhe-ffdhe2048, ...
TLS_DHE_RSA_WITH_AES_256_CCM	dhe-ffdhe2048, ...
TLS_DHE_PSK_WITH_AES_128_CCM	psk-dhe-ffdhe2048, ...
TLS_DHE_PSK_WITH_AES_256_CCM	psk-dhe-ffdhe2048, ...
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	ecdhe-secp256r1, ...
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	ecdhe-secp256r1, ...
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	dhe-ffdhe2048, ...
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	psk-ecdhe-secp256r1,...
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	psk-dhe-ffdhe2048, ...
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	psk-ecdhe-secp256r1,...
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	psk-ecdhe-secp256r1,...
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	psk-ecdhe-secp256r1,...

Table 1-5 TLS 1.2 Compatibility Matrix Part 5: ciper-suites mapping to key-negotiation-algorithm

ciper-suites in hello -params-grouping	HASH
TLS_AES_128_GCM_SHA256	sha-256
TLS_AES_256_GCM_SHA384	sha-384
TLS_CHACHA20_POLY1305_SHA256	sha-256
TLS_AES_128_CCM_SHA256	sha-256

Table 2-1 TLS 1.3 Compatibility Matrix Part 1: ciper-suites mapping to hash-algorithm

ciper-suites in hello -params-grouping	symmetric
TLS_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_AES_256_GCM_SHA384	enc-aes-128-gcm
TLS_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_AES_128_CCM_SHA256	enc-aes-128-ccm

Table 2-2 TLS 1.3 Compatibility Matrix Part 2: ciper-suites mapping to symmetric-key--encryption-algorithm

ciper-suites in hello -params-grouping	symmetric
TLS_AES_128_GCM_SHA256	mac-aes-128-gcm
TLS_AES_256_GCM_SHA384	mac-aes-128-gcm
TLS_CHACHA20_POLY1305_SHA256	mac-chacha20-poly1305
TLS_AES_128_CCM_SHA256	mac-aes-128-ccm

Table 2-3 TLS 1.3 Compatibility Matrix Part 3: ciper-suites mapping to MAC-algorithm

signatureScheme in hello -params-grouping	signature
rsa-pkcs1-sha256	rsa-pkcs1-sha256
rsa-pkcs1-sha384	rsa-pkcs1-sha384
rsa-pkcs1-sha512	rsa-pkcs1-sha512
rsa-pss-rsae-sha256	rsa-pss-rsae-sha256
rsa-pss-rsae-sha384	rsa-pss-rsae-sha384
rsa-pss-rsae-sha512	rsa-pss-rsae-sha512
rsa-pss-pss-sha256	rsa-pss-pss-sha256
rsa-pss-pss-sha384	rsa-pss-pss-sha384
rsa-pss-pss-sha512	rsa-pss-pss-sha512
ecdsa-secp256r1-sha256	ecdsa-secp256r1-sha256
ecdsa-secp384r1-sha384	ecdsa-secp384r1-sha384
ecdsa-secp521r1-sha512	ecdsa-secp521r1-sha512
ed25519	ed25519
ed448	ed448

Table 2-4 TLS 1.3 Compatibility Matrix Part 4: SignatureScheme mapping to signature-algorithm

supported Groups in hello -params-grouping	key-negotiation
dhe-ffdhe2048	dhe-ffdhe2048
dhe-ffdhe3072	dhe-ffdhe3072
dhe-ffdhe4096	dhe-ffdhe4096
dhe-ffdhe6144	dhe-ffdhe6144
dhe-ffdhe8192	dhe-ffdhe8192
psk-dhe-ffdhe2048	psk-dhe-ffdhe2048
psk-dhe-ffdhe3072	psk-dhe-ffdhe3072
psk-dhe-ffdhe4096	psk-dhe-ffdhe4096
psk-dhe-ffdhe6144	psk-dhe-ffdhe6144
psk-dhe-ffdhe8192	psk-dhe-ffdhe8192
ecdhe-secp256r1	ecdhe-secp256r1
ecdhe-secp384r1	ecdhe-secp384r1
ecdhe-secp521r1	ecdhe-secp521r1
ecdhe-x25519	ecdhe-x25519
ecdhe-x448	ecdhe-x448
psk-ecdhe-secp256r1	psk-ecdhe-secp256r1
psk-ecdhe-secp384r1	psk-ecdhe-secp384r1
psk-ecdhe-secp521r1	psk-ecdhe-secp521r1
psk-ecdhe-x25519	psk-ecdhe-x25519
psk-ecdhe-x448	psk-ecdhe-x448

Table 2-5 TLS 1.3 Compatibility Matrix Part 5: Supported Groups mapping to key-negotiation-algorithm

Note that in Table 1-5:

- dhe-ffdhe2048, ... is the abbreviation of dhe-ffdhe2048, dhe-ffdhe3072, dhe-ffdhe4096, dhe-ffdhe6144, dhe-ffdhe8192;

- psk-dhe-ffdhe2048, ... is the abbreviation of psk-dhe-ffdhe2048, psk-dhe-ffdhe3072, psk-dhe-ffdhe4096, psk-dhe-ffdhe6144, psk-dhe-ffdhe8192;

- ecdhe-secp256r1, ... is the abbreviation of ecdhe-secp256r1, ecdhe-secp384r1, ecdhe-secp521r1, ecdhe-x25519, ecdhe-x448;

- psk-ecdhe-secp256r1, ... is the abbreviation of psk-ecdhe-secp256r1, psk-ecdhe-secp384r1, psk-ecdhe-secp521r1, psk-ecdhe-x25519, psk-ecdhe-x448.

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.

5.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-tls-common" module.

```
module: ietf-tls-common

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

5.2. Example Usage

This section shows how it would appear if the transport-params-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>
```

5.3. 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@2020-03-08.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) 2019 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 XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); 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 2020-03-08 {
       description
         "Initial version";
       reference
         "RFC XXXX: 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 XXXX for
             valid combinations.

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

<CODE ENDS>
```

6. Security Considerations

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

There are a number of data nodes defined in the YANG modules that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

*: The entire subtree defined by the grouping statement in both the "ietf-ssh-client" and "ietf-ssh-server" modules is sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., or even the modification of transport or keepalive parameters can dramatically alter the implemented security

policy. For this reason, this node is protected the NACM extension "default-deny-write".

Some of the readable data nodes in the YANG modules may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/tls-client-parameters/client-identity/: This subtree in the "ietf-tls-client" module contains nodes that are additionally sensitive to read operations such that, in normal use cases, they should never be returned to a client. Some of these nodes (i.e., public-key/local-definition/private-key and certificate/local-definition/private-key) are already protected by the NACM extension "default-deny-all" set in the "grouping" statements defined in [I-D.ietf-netconf-crypto-types].

/tls-server-parameters/server-identity/: This subtree in the "ietf-tls-server" module contains nodes that are additionally sensitive to read operations such that, in normal use cases, they should never be returned to a client. All of these nodes (i.e., host-key/public-key/local-definition/private-key and host-key/certificate/local-definition/private-key) are already protected by the NACM extension "default-deny-all" set in the "grouping" statements defined in [I-D.ietf-netconf-crypto-types].

Some of the operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

*: The groupings defined in this document include "action" statements that come from groupings defined in [I-D.ietf-netconf-crypto-types]. Please consult that document for the security considerations of the "action" statements defined by the "grouping" statements defined in this document.

7. IANA Considerations

7.1. 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-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.

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.
```

7.2. 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-client
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-client
prefix:       tlsc
reference:    RFC XXXX

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

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

8. References

8.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K. and H. Wang, "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-13 (work in progress), November 2019.

[I-D.ietf-netconf-keystore]
Watsen, K., "A YANG Data Model for a Keystore", draft-ietf-netconf-keystore-15 (work in progress), November 2019.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "A YANG Data Model for a Truststore", draft-ietf-netconf-trust-anchors-08 (work in progress), November 2019.

[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>.

8.2. Informative References

[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>.

Appendix A. Change Log

A.1. 00 to 01

- o Noted that '0.0.0.0' and '::' might have special meanings.
- o Renamed "keychain" to "keystore".

A.2. 01 to 02

- o Removed the groupings containing transport-level configuration. Now modules contain only the transport-independent groupings.
- o Filled in previously incomplete 'ietf-tls-client' module.
- o Added cipher suites for various algorithms into new 'ietf-tls-common' module.

A.3. 02 to 03

- o Added a 'must' statement to container 'server-auth' asserting that at least one of the various auth mechanisms must be specified.
- o Fixed description statement for leaf 'trusted-ca-certs'.

A.4. 03 to 04

- o Updated title to "YANG Groupings for TLS Clients and TLS Servers"
- o Updated leafref paths to point to new keystore path
- o Changed the YANG prefix for ietf-tls-common from 'tlscom' to 'tlscmn'.
- o Added TLS protocol verions 1.0 and 1.1.
- o Made author lists consistent
- o Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- o Updated YANG to use typedefs around leafrefs to common keystore paths
- o Now inlines key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

- o Merged changes from co-author.

A.6. 05 to 06

- o Updated to use trust anchors from trust-anchors draft (was keystore draft)
- o Now Uses new keystore grouping enabling asymmetric key to be either locally defined or a reference to the keystore.

A.7. 06 to 07

- o factored the tls-[client|server]-groupings into more reusable groupings.
- o added if-feature statements for the new "x509-certificates" feature defined in draft-ietf-netconf-trust-anchors.

A.8. 07 to 08

- o Added a number of compatibility matrices to Section 5 (thanks Frank!)
- o Clarified that any configured "cipher-suite" values need to be compatible with the configured private key.

A.9. 08 to 09

- o Updated examples to reflect update to groupings defined in the keystore draft.
- o Add TLS keepalives features and groupings.
- o Prefixed top-level TLS grouping nodes with 'tls-' and support mashups.
- o Updated copyright date, boilerplate template, affiliation, and folding algorithm.

A.10. 09 to 10

- o Reformatted the YANG modules.

A.11. 10 to 11

- o Collapsed all the inner groupings into the top-level grouping.
- o Added a top-level "demux container" inside the top-level grouping.
- o Added NACM statements and updated the Security Considerations section.
- o 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.
- o Updated the boilerplate text in module-level "description" statement to match copyeditor convention.

A.12. 11 to 12

- o In server model, made 'client-authentication' a 'presence' node indicating that the server supports client authentication.
- o In the server model, added a 'required-or-optional' choice to 'client-authentication' to better support protocols such as RESTCONF.
- o 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".
- o 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.
- o Fixed a couple references (section 2 --> section 3)

A.13. 12 to 13

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

A.14. 12 to 13

- o Removed 'container' under 'client-identity' to match server model.
- o Updated examples to reflect change grouping in keystore module.

A.15. 13 to 14

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

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

A.16. 14 to 15

- o 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

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

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

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

A.18. 16 to 17

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

- o Removed choice required-or-optional.

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

- o 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

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

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

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

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

- o Updated examples to reflect new "bag" addition to truststore.

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

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

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

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

Acknowledgements

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

Authors' Addresses

Kent Watsen
Watsen Networks

EMail: kent+ietf@watsen.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com

Liang Xia
Huawei

EMail: frank.xialiang@huawei.com