NETCONF Server and RESTCONF Server Configuration Models
draft-ietf-netconf-server-model-09

Abstract

This draft defines a NETCONF server configuration data model and a RESTCONF server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, call-home parameters, client authentication, and related parameters.

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. Please note that no other RFC Editor instructions are specified anywhere else 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:

- draft-ietf-netconf-restconf
- draft-ietf-netconf-call-home
- draft-ietf-rtgwg-yang-key-chain

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

- "VVVV" --> the assigned RFC value for this draft
- "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-call-home
Artwork in this document contains placeholder values for ports pending IANA assignment from "draft-ietf-netconf-call-home". Please apply the following replacements:

- "7777" --> the assigned port value for "netconf-ch-ssh"
- "8888" --> the assigned port value for "netconf-ch-tls"
- "9999" --> the assigned port value for "restconf-ch-tls"

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

- "2016-03-16" --> the publication date of this draft

The following two Appendix sections are to be removed prior to publication:

- Appendix A. Change Log
- Appendix B. Open Issues

Artwork in the document contains a temporary YANG containers that need to be removed.

- The "listening-ssh-server" container listed at the end of the artwork in Section 4.2.3 needs to be removed. Please remove the ten lines starting with "container listening-ssh-server (" and ending with ")".
- The "listening-tls-server" container listed at the end of the artwork in Section 4.3.3 needs to be removed. Please remove the ten lines starting with "container listening-tls-server (" and ending with ")".

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 http://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 17, 2016.

Copyright Notice

Copyright (c) 2016 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 (http://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 ............................................. 4
   1.1. Terminology .......................................... 5
   1.2. Tree Diagrams ....................................... 5
2. Objectives ............................................... 5
   2.1. Support all NETCONF and RESTCONF transports .......... 5
   2.2. Enable each transport to select which keys to use .... 6
   2.3. Support authenticating NETCONF/RESTCONF clients certificates ............................................. 6
   2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames ............................. 6
   2.5. Support both listening for connections and call home .. 6
   2.6. For Call Home connections ............................. 6
      2.6.1. Support more than one NETCONF/RESTCONF client .. 7
      2.6.2. Support NETCONF/RESTCONF clients having more than one endpoint ..................................... 7
      2.6.3. Support a reconnection strategy ...................... 7
      2.6.4. Support both persistent and periodic connections .. 7
      2.6.5. Reconnection strategy for periodic connections .......... 7
      2.6.6. Keep-alives for persistent connections ............... 8
      2.6.7. Customizations for periodic connections ............. 8
3. High-Level Design ......................................... 8
4. Solution .................................................. 9
   4.1. The System Keychain Model ............................ 9
      4.1.1. Tree Diagram ...................................... 9
      4.1.2. Example Usage ................................... 10
      4.1.3. YANG Model ..................................... 18
1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model and a RESTCONF [draft-ietf-netconf-restconf] server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, call-home parameters, client authentication, and related parameters.
1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":")
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Objectives

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF and RESTCONF services on a network element. This scope includes the following objectives:

2.1. Support all NETCONF and RESTCONF transports

The YANG module should support all current NETCONF and RESTCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [RFC7589], and RESTCONF over TLS [draft-ietf-netconf-restconf], and to be extensible to support future transports as necessary.

Because implementations may not support all transports, the module should use YANG "feature" statements so that implementations can accurately advertise which transports are supported.
2.2. Enable each transport to select which keys to use

Servers may have a multiplicity of host-keys or server-certificates from which subsets may be selected for specific uses. For instance, a NETCONF server may want to use one set of SSH host-keys when listening on port 830, and a different set of SSH host-keys when calling home. The data models provided herein should enable configuration of which keys to use on a per-use basis.

2.3. Support authenticating NETCONF/RESTCONF clients certificates

When a certificate is used to authenticate a NETCONF or RESTCONF client, there is a need to configure the server to know how to authenticate the certificates. The server should be able to authenticate the client’s certificate either by using path-validation to a configured trust anchor or by matching the client-certificate to one previously configured.

2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames

When a client certificate is used for TLS client authentication, the NETCONF/RESTCONF server must be able to derive a username from the authenticated certificate. Thus the modules defined herein should enable this mapping to be configured.

2.5. Support both listening for connections and call home

The NETCONF and RESTCONF protocols were originally defined as having the server opening a port to listen for client connections. More recently the NETCONF working group defined support for call-home ([draft-ietf-netconf-call-home]), enabling the server to initiate the connection to the client, for both the NETCONF and RESTCONF protocols. Thus the modules defined herein should enable configuration for both listening for connections and calling home. Because implementations may not support both listening for connections and calling home, YANG "feature" statements should be used so that implementation can accurately advertise the connection types it supports.

2.6. For Call Home connections

The following objectives only pertain to call home connections.
2.6.1. Support more than one NETCONF/RESTCONF client

A NETCONF/RESTCONF server may be managed by more than one NETCONF/RESTCONF client. For instance, a deployment may have one client for provisioning and another for fault monitoring. Therefore, when it is desired for a server to initiate call home connections, it should be able to do so to more than one client.

2.6.2. Support NETCONF/RESTCONF clients having more than one endpoint

An NETCONF/RESTCONF client managing a NETCONF/RESTCONF server may implement a high-availability strategy employing a multiplicity of active and/or passive endpoint. Therefore, when it is desired for a server to initiate call home connections, it should be able to connect to any of the client’s endpoints.

2.6.3. Support a reconnection strategy

Assuming a NETCONF/RESTCONF client has more than one endpoint, then it becomes necessary to configure how a NETCONF/RESTCONF server should reconnect to the client should it lose its connection to one the client’s endpoints. For instance, the NETCONF/RESTCONF server may start with first endpoint defined in a user-ordered list of endpoints or with the last endpoints it was connected to.

2.6.4. Support both persistent and periodic connections

NETCONF/RESTCONF clients may vary greatly on how frequently they need to interact with a NETCONF/RESTCONF server, how responsive interactions need to be, and how many simultaneous connections they can support. Some clients may need a persistent connection to servers to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for server to initiate connections, it should be configurable if the connection is persistent or periodic.

2.6.5. Reconnection strategy for periodic connections

The reconnection strategy should apply to both persistent and periodic connections. How it applies to periodic connections becomes clear when considering that a periodic "connection" is a logical connection to a single server. That is, the periods of unconnectedness are intentional as opposed to due to external reasons. A periodic "connection" should always reconnect to the same server until it is no longer able to, at which time the reconnection strategy guides how to connect to another server.
2.6.6. Keep-alives for persistent connections

If a persistent connection is desired, it is the responsibility of the connection initiator to actively test the "aliveness" of the connection. The connection initiator must immediately work to reestablish a persistent connection as soon as the connection is lost. How often the connection should be tested is driven by NETCONF/RESTCONF client requirements, and therefore keep-alive settings should be configurable on a per-client basis.

2.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF/RESTCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF/RESTCONF client may have to wait to send data to a server. A server may connect to a client before this interval expires if desired (e.g., to send data to a client).

3. High-Level Design

The solution presented in this document defines a configurable keychain object, reusable groupings for SSH and TLS based servers, and, finally, the configurable NETCONF and RESTCONF server objects, which are the primary purpose for this draft. Each of these are defined in a distinct YANG module, thus a total of five YANG modules are defined in this document. The relationship between these five YANG modules is illustrated by the tree diagram below.

```
+--------------------+
 |ietf-system-keychain|
+--------------------+

  ^       ^
 <leafref>  <leafref>

  +-------------------+  +------------------+
 |                   |  |                   |
 |<uses>             |  |<augments>         |
 |                   |  |                   |
 +-------------------+  +------------------+

+-------------------+  +------------------+
 |ietf-ssh-server|  |ietf-tls-server  |
+-------------------+  +------------------+

  ^       ^
 <leafref>  <leafref>

  +-------------------+  +-------------------+
 |                    |  |                    |
 |<augments>           |  |<augments>         |
 |                    |  |                   |
 +-------------------+  +-------------------+

+-------------------+  +-------------------+
 |ietf-netconf-server|  |ietf-restconf-server|
+-------------------+  +-------------------+
```
4. Solution

Each of the following five sections relate to one of the YANG modules depicted by the figure above.

4.1. The System Keychain Model

The system keychain model defined in this section provides a configurable object having the following characteristics:

- A semi-configurable list of private keys, each with one or more associated certificates. Private keys MUST be either preinstalled (e.g., an IDevID key), be generated by request, or be loaded by request. Each private key is MAY have associated certificates, either preinstalled or configured after creation.

- A configurable list of lists of trust anchor certificates. This enables the server to have use-specific trust anchors. For instance, one list of trust anchors might be used to authenticate management connections (e.g., client certificate-based authentication for NETCONF or RESTCONF connections), and a different list of trust anchors might be used for when connecting to a specific Internet-based service (e.g., a zero touch bootstrap server).

- An RPC to generate a certificate signing request for an existing private key, a passed subject, and an optional attributes. The signed certificate returned from an external certificate authority (CA) can be later set using a standard configuration change request (e.g., <edit-config>).

- An RPC to request the server to generate a new private key using the specified algorithm and key length.

- An RPC to request the server to load a new private key.

4.1.1. Tree Diagram
module: ietf-system-keychain
  +--rw keychain
    +--rw private-keys
      +--rw private-key* [name]
        +--rw name string
        +--ro algorithm? kc:algorithms
        +--ro key-length? uint32
        +--ro public-key binary
        +--rw certificate-chains
          +--rw certificate-chain* [name]
            +--rw name string
            +--rw certificate* binary
          +--x generate-certificate-signing-request
            +--w input
            |   +--w subject binary
            |   +--ro attributes? binary
            +--ro output
              +--ro certificate-signing-request binary
          +--x generate-private-key
            +--w input
            |   +--w name string
            |   +--w key-usage? enumeration
            |   +--w algorithm kc:algorithms
            |   +--w key-length? uint32
          +--x load-private-key
            +--w input
            |   +--w name string
            |   +--w private-key binary
        +--rw trusted-certificates* [name]
          +--rw name string
          +--rw description? string
          +--rw trusted-certificate* [name]
          +--rw name string
          +--rw certificate? binary

notifications:
  +--n certificate-expiration
    +--ro certificate instance-identifier
    +--ro expiration-date yang:date-and-time

4.1.2. Example Usage

The following example illustrates the "generate-private-key" action in use with the RESTCONF protocol and JSON encoding.
The following example illustrates the "load-private-key" action in use with the RESTCONF protocol and JSON encoding.
The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol.

REQUEST
-------

```xml
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <keychain
xmlns="urn:ietf:params:xml:ns:ietf-system-keychain">
      <name>ex-key-sect571r1</name>
      <private-key>
        NGcEk3UE90cnNvIGlud25GL015R315MC5LbmljZS5G
      </private-key>
    </keychain>
  </action>
</rpc>
```

RESPONSE
---------

HTTP/1.1 204 No Content
Date: Mon, 31 Oct 2015 11:01:00 GMT
Server: example-server

Watsen & Schoenwaelder Expires September 17, 2016
<private-keys>
<private-key>
    <name>ex-key-sect571r1</name>
    <generate-certificate-signing-request>
        <subject>
            cztvaWRoc2RmZ2tqaHNkZmdramRzZnZgTmam5idnNvO2R
            manZvO3NkZmJpdmhzZGZpbHVidtvc21kZmhidm1bHN1mO
            Z2aXXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmR6Zgo=
        </subject>
        <attributes>
            bwtakWRoc2RmZ2tqaHNkZmdramRzZnZgTmam5idnNvut4
            arnZvO3NkZmJpdmhzZGZpbHVidtvc21kZmhidm1bHNkYm
            Z2aXXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmC6Rhp=
        </attributes>
    </generate-certificate-signing-request>
</private-key>
</private-keys>
</keychain>
</action>
</rpc>

RESPONSE
--------

<?xml version="1.0" encoding="UTF-8"?>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQW5Z
    0F3SUJBZ01kQUpRT2t3bGpNK2pjtUUEwR0NTcUdTSW1rRFFWQkJRVU
    FNRFF4Q3pBSk5Jn1YKqfZVEfSv1RNukF3RGd2RFZRUUtFd2RsZUd
    GDGNHeGxNuk13RFC2RFZRUURfd3BEWt3Z1NYTnpkV1Z5TU10WAPe
    d1R1V4XyPB5rJn12Q0UQ2tOU1RDkpjM04xW1hjd2daOHDtEUtV
    K29aSWHzYO4QkVENkFkJRURnWTBBS1HSmkFwR0JBTXVZmFNEV3
    ELl1QWMrQ1RsTkNncm0c6Ew1Um5ydxZsOFRICuTjgZQY3N02k1K1T1
    FaNzlnNLNWlmsMldaHE1bUVickJNNitGNzdjTAvU25FcFE0TnV
    bXBDT2YKQwDQkFR2pynXy3Z2Fr0hRWURWuVBPQkJZRUZKY1o2W
    UiR01PND4a1jnPB3jtREdsRUNCVTFNR1HFQTFVZApJd1JKUZ1QU
    ZKY1o2WUriri01PND4a1jnPB3jtREdsRUNCVTFVGlrTmBMЕ1R53d
    mMKTEU0QExVRVd0VCL3dRRUF3SUNREFTQmdOvhSTUJBzjFQQO
    RR0FRCSC9B20BTVwR0NTcUdTSW1rFFQgCUVQOQTRHQkFMMnx
    rWmFGW5cyaGRMhNhZnZpbnBneHA4egU0SHRhbStadHPzaF1S3Bx
    T3p4YXJCbFPdSH1LCk1Vbc9GVzrVtIQS1VDeETF40NQY2zmk2d
    c40tSS5IyLYWLO0p7m1Q50VXSTF4K1IaDzma5crQzQ1QXg1RNW
    SWHzgZjdVM2xZgotLS0tLVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  </certificate-signing-request>
</rpc-reply>
The following example illustrates what a fully configured keychain object might look like. The private-key shown below is consistent with the generate-private-key and generate-certificate-signing-request examples above. This example also assumes that the resulting CA-signed certificate has been backed up on the server. Lastly, this example shows that three lists of trusted certificates have been configured.

```xml
<keychain xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">
  <!-- private keys and associated certificates --
  <private-keys>
    <private-key>
      <name>tpm-protected-key</name>
      <algorithm>sect571r1</algorithm>
      <public-key>
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZGtmam5idnNvO2RmanZvO3NkZ
        mJpdmhZ2pHVidjtvc2ikZmhBm11bHNkYmZ2aXN1ZGZpYmhzZG872m
        Jv03NkZ25iO29pLmR6Zgo=
      </public-key>
      <certificate-chains>
        <certificate-chain>
          <name>default-idev-switch</name>
          <certificate>
            diR1V4RXpBUkJnT1ZCQ1UQ2tOU1RDQkpjM04xW1hJd2da0HdEUVVL
            LS0tLS1CRUdJTXIBDRVJUSU3Q0FURS0tLS0tCk1JSUNrekNDQWZ5Z
            KS29aSWWh2Y04KQVFFQkJRQRnWTBTBU1HSkFr0JBTXVv2mFPEVE3
            0F3SUJBZ01kKQupRT2t3bGpNKN2pjTUEwRONTc0dTSWiZrrFFQkJRUV
            FNRFFQ43pBSkJnT1YQkFZVEFsv1RNUkF3RGd2RFZRUUtFD2RsZUd
            GdGkGNeGxNuk13RVFZFR2URUFRd3BEWt3Z1NYTnpkV1ZSTU0WApeE
            ZKY02WURiR01PND8a4J1P3tJtErSdRnCvTFvYVGrTmpBME1Rcd
            mMKTUE0RExYWRU0vC1d3dRRUF3SUNCREFTQmdovkhSTIJZBHjFQ0
            RBB0RFS9C2ZOBvTEwRONTc0dTSWiZrrFFQgpUUVBQTHQkFMMnx
            rWmFGNWcyaGR6MWnh2z2PbnBNeA4eG0oSRRhBStadHpLazFL3SBx
            Xp4YXCbPdSH1Lck1VbC9GVzrVlVRS1VDeEtFTE40NEY2yzmk2d
            c4d0tSSElkYW1WLOpGtmLQS0VXSTF4K1i1aDzmarcrQzQ1QXg1RW
            SWMQ2xTgoTtS0tLUVORCBDRVJUSU3Q0FURS0tLS0tCg==
          </certificate>
        </certificate-chain>
        <certificate-chain>
          KS29aSWWh2Y04KQVFFQkJRQRnWTBTBU1HSkFr0JBTXVv2mFPEVE3
          E1iQWmQ1RsTkNmc0d6cEw1U5ydXZsORFcUJTdGZQY3N0Zk1K1T
          FaNz1n1NLN1Wl1sMldzaHE1BUViCkJKNItnGNzdjbTAuVUZ5FCFE0TVn
          bXBE2T2YQaDQkFBRG2pnXYd3Z2Fr0dHRWURU1jBPQkJZRUZKY1o2W
          LS0tLS1CRUdJTXIBDRVJUSQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
          0F3SUJBZ01kKQupRT2t3bGpNKN2pjTUEwRONTc0dTSWiZrrFFQkJRUV
          FNRFFQ43pBSkJnT1YQkFZVEFsv1RNUkF3RGd2RFZRUUtFD2RsZUd
          GdGkGNeGxNuk13RVFZFR2URUFRd3BEWt3Z1NYTnpkV1ZSTU0WApeE
          diR1V4RXpBUkJnT1ZCQ1UQ2tOU1RDQkpjM04xW1hJd2da0HdEUVVL
        </certificate-chain>
      </certificate-chains>
    </private-key>
  </private-keys>
</keychain>
```
URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTpFVZApJd1JkTUZ1QU RBROFRSC9BZ0VBUTUEwRONTcudTSWIzRFFQgCuVUvBqTRHQkFMMmx rWmFGNWycaGr6MNWnzh2PbnBneHA4eG00SRHbhStAdHplAzF13S3X c4d0tSSElkyYW1WL0pGmlqMQS0VXSTF4K11aDmazcrQzQ1QXg1RZW SVSUZQ0FURS07LS0tCg==
</certificate>
</certificate-chain>

<certificate-chain>
  <name>my-ldevid-chain</name>
  <certificate>
    0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwRONTcudTSWIzRFFQkJRUV FNRFFQ4pBSKJnTYIYQkFZVEFsV1RNukF3RGdZRFZRUUFdF2DrS2Ud GdGNHegXnuK13RVEFZRFZRRUURFD3BEWt3Z1NYTnpvK1Z5TUI0WAPE diR1V4RXpBUkJnTIzCQ1UQ2tOU1RDQkpjM04xW1hJd2daOHDhEUvL LS0tLS1CR0UdJT1iBDRTVJUSUZQ0FURS07LS0tCk1JSUNrekJNDQWZ5Z KS29aSwH2Y04KQVFQkJRQRnnWTBTTU1HSKFrR0JBTXVzvMFPNVE3 ElI1QWMrQ1RsTkNmc0d6cEw1Um5yXZsOFRIcUJTdGQZ3NOZk1KTl FaNz1nNLNvVldsMldzAE1bU0vCkjNNItGNzdjbTAvU25FyE0F0nVn ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvG1rTmpeBEM1Rc3d mKsTUE0R0ExVWREd0VCI3dRRUF3SUNCREFTQmdOVkhSTUBZJhFQ0 RBROFRSC9BZ0VBUTUEwRONTcudTSWIzRFFQgCuVUvBqTRHQkFMMmx rWmFGNWycaGr6MNWnzh2PbnBneHA4eG00SRHbhStAdHplAzF13S3X Txp4YXJCbfpDShlLcK1v1bC9GVzRtV1RQs1VDeEtFE40NExY2Zm2k2 c4d0tSSElklcyYW1wL0pGmlqMQS0VXSTF4K11aDmazcrQzQ1QXg1RZW SWM2xZTgotLS0tLUvORCBdRTVJUSUZQ0FURS07LS0tCg==</certificate>
</certificate-chain>

<private-key>LS0tLS1CRL0dJTIiBDRTVJUSUZQ0FURS07LS0tCk1JSUNrekJNDQWZ5Z 0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwRONTcudTSWIzRFFQkJRUV FNRFFQ4pBSKJnTYIYQkFZVEFsV1RNukF3RGdZRFZRUUFdF2DrS2Ud GdGNHegXnuK13RVEFZRFZRRUURFD3BEWt3Z1NYTnpvK1Z5TUI0WAPE diR1V4RXpBUkJnTIzCQ1UQ2tOU1RDQkpjM04xW1hJd2daOHDhEUvL KS29aSwH2Y04KQVFQkJRQRnnWTBTTU1HSKFrR0JBTXVzvMFPNVE3 ElI1QWMrQ1RsTkNmcc0d6cEw1Um5yXZsOFRIcUJTdGQZ3NOZk1KTl FaNz1nNLNvVldsMldzAE1bU0vCkjNNItGNzdjbTAvU25FyE0F0nVnbXBDRZTYQkQdNQkFRBR2pynXz32Fr0hRWURWjBPQkjrZURUKY1o2W URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTpFVZApJd1JkTUZ1QU ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvG1rTmpeBEM1Rc3d mKsTUE0R0ExVWREd0VCI3dRRUF3SUNCREFTQmdOVkhSTUBZJhFQ0 RBROFRSC9BZ0VBUTUEwRONTcudTSWIzRFFQgCuVUvBqTRHQkFMMmx rWmFGNWycaGr6MNWnzh2PbnBneHA4eG00SRHbhStAdHplAzF13S3X Txp4YXJCbfpDShlLcK1v1bC9GVzRtV1RQs1VDeEtFE40NExY2Zm2k2 c4d0tSSElklcyYW1wL0pGmlqMQS0VXSTF4K11aDmazcrQzQ1QXg1RZW SWHgz2jdVM2xZTgotLS0tLUvORCBdRTVJUSUZQ0FURS07LS0tCg==</private-key>
<trusted-certificates>
  <name>explicitly-trusted-client-certs</name>
  <description>
  Specific client authentication certificates that are to be explicitly trusted NETCONF/RESTCONF clients. These are needed for client certificates not signed by our CA.
  </description>
</trusted-certificates>

<trusted-certificate>
  <name>George Jetson</name>
  <certificate>
  QmdOVkJBWVRBbFZUTVJBd0RnWURUVFLRXdkbAp1R020Y0d4bE1RNHzdEQ
  MkF6a3hqUD1VQWtHR0dvS1U1eUc1SVR0Wm0vK3BR2FieXVDMjBRd2kvZ
  25PznZENhONApXY0TaUpZK2xtYW3s3TRORUZXZ9RdGp4NU1XZmdvN2
  RV0JCU2t2MNHFhEUFVwpSmYyOtwXbUUNOEo5akRqMdvXkTHUUY
  VEJiZ0JTWdibUEKmhpRH50VkvVHFLNwd4cFJBZ120YUU0cERZd05ER
  UXnQWhtHoTFVUjoUNWvk14URuBT0JnT12CQ9W9UQjWJWAp2VzF3YdVdE
  V6QVJCZ05WqFKFNVENsT1NU0QK7yZnNOMyVpYSONDUUNVRHBNs116UG8
  RNFQmdOVhktSUJbZjhFCkFqQUFQRHRHQTFVZER3RU1vd1FQXdJSGdQnBc
  Z05WpFI4RLqQmdNRj3nSXFZB2hoNW9kSFJ3T2k4d1plYaGgK1hC
  WiPijMiB2WlhoaGJYQnNaUzVqY215a9LUUTJNRFF4QpBSkJnT12CQV1UQw
  xWXE15QXeDI11EVI1FRSwpD2s2zUDgdGNHeGxNkU13RVF3RFZRUURFd4
  EVWt3Z1NYtpkV125UWeRo0NTcUdTSWizRFFFFkNJVRUFNEdCCkFCc3BK
  WmdsK2gyTj3qMtoGMjWhB1cDFVaWc3OGrRkYyRTrFwSt4ZVRbVFVM
  TQzjC1S3j0MlFQlzV5eGUKNQxMdxCV0dxuUjurbE15N0Y1L2Ika2M4al
  zSNWsdwVXBCYn4dmtNanFtZjJna3RqZHBxeFppUUtTbndWZTF22wot
  LS0tLUVOCRDRVJUSUZQFCU0S0tLS0tCg==
  </certificate>
</trusted-certificate>

<trusted-certificates>
  <name>Fred Flintstone</name>
  <certificate>
  V1EV1FRREEV3m9ZWEJ3ZVRDQm56QU5CZ2txtaGtpPrZ13EMEJBUUVGQUFQP
  pQXdnWwYwCmC2ZFrZaS1p2bd1XTW44eUHy2MhoBUFRaUhVUVZ
  rUrUpQy9hSFA3eGJXQWiIa054ZStaU2hr2nBsL3UKhVhsTjHzSUDI0hH
  NGEc3UE90cnVFVrJTNud8VBOWFQ0FSSXDezn0VPCk1CMEDVBMVRGkRd
  VEJiZ0JTWdibUEKmhpRH50VkvVHFLNwd4cFJBZ120YUU0cERZd05ER
  V6QVJCZ05WqFKFNVENsT1NU0QK7yZnNOMyVpYSONDUUNVRHBNs116UG8
  RNFQmdOVhktSUJbZjhFCkFqQUFQRHRHQTFVZER3RU1vd1FQXdJSGdQnBc
  Z05WpFI4RLqQmdNRj3nSXFZB2hoNW9kSFJ3T2k4d1plYaGgK1hC
  WiPijMiB2WlhoaGJYQnNaUzVqY215a9LUUTJNRFF4QpBSkJnT12CQV1UQw
  xWXE15QXeDI11EVI1FRSwpD2s2zUDgdGNHeGxNkU13RVF3RFZRUURFd4
  EVWt3Z1NYtpkV125UWeRo0NTcUdTSWizRFFFFkNJVRUFNEdCCkFCc3BK
  WmdsK2gyTj3qMtoGMjWhB1cDFVaWc3OGrRkYyRTrFwSt4ZVRbVFVM
  TQzjC1S3j0MlFQlzV5eGUKNQxMdxCV0dxuUjurbE15N0Y1L2Ika2M4al
  zSNWsdwVXBCYn4dmtNanFtZjJna3RqZHBxeFppUUtTbndWZTF22wot
  LS0tLUVOCRDRVJUSUZQFCU0S0tLS0tCg==
  </certificate>
</trusted-certificate>
<trusted-certificates>
  <name>deployment-specific-ca-certs</name>
  <description>
  Trust anchors used only to authenticate NETCONF/RESTCONF client connections. Since our security policy only allows authentication for clients having a certificate signed by our CA, we only configure its certificate below.
  </description>
  <trusted-certificate>
    <name>ca.example.com</name>
    <certificate>
      WMdsK2gyTTg3QmtGMjJhWbW1CdFFVAcC30EGrRkYyRTFwdSt4ZVRJbVFFMjLQL1sWp0Cf7Mn6RLR05EMu20VJpK2FWNw2NtDznC4tadVjmZgB8YjkzSfnWSdWwVXCYaYnA4dmtNanFtZj3ma3RgZHXeFppUtbnTWZTF2ZwotNGcEk3UE90cnNFVjJwTUNb0VbQWFPQ0FSSXdn20VPCk1CMEqRMVbUKGd
      VEjJi20JTWEmlbUEKmhpRVHvOTkVvVHFLNw4cfFJB210YUU0CERz05ER
      V6QVJCC05WTQFNvXrT1NUQ0JkYzNOMvPxY0NDUUMRHBNSl16UG8zREF
      NQmdOVkhSTUJZ2jhFCkFqOQFQNRQHFZTVJZER3RU1vdfpQXxJSGdEQnBC
      Z05WSF14Rv1qQmdNRJznSXFBZ2hoNW9kSFJ3I2k4d1pYaGqYljC1pTN
      WpiJMB2W1hoa4GJYQnNaUzVqYr215aU9LTJNNFFQ3pBSkJnT1ZCQV1UQWQmdOVkJBVWRBBf2UTVJBD0rWURUWVFRLxdkbAp1R0Z0Y0d4Be1RNHHEQ
      MKF6a3hQUd1QWHR0dSV1U1eUc1SR0Wm0vK3B0R2F1eXVDMjBRd2kVZ
      25P3npZNEhONApXY0pTa0pZEK2xtYWw3RTRORUZXS9RdGp4NLU1ZXmdvN2
      RJSUQFRRStS0Cg==
    </certificate>
  </trusted-certificate>
</trusted-certificates>

<!-- trust anchors for random HTTPS servers on Internet -->
<trusted-certificates>
  <name>common-ca-certs</name>
  <description>
  Trusted certificates to authenticate common HTTPS servers. These certificates are similar to those that might be shipped with a web browser.
  </description>
</trusted-certificates>

<!-- trust anchors for netconf/restconf clients -->
<trusted-certificates>
  <name>deployment-specific-ca-certs</name>
  <description>
  Trust anchors used only to authenticate NETCONF/RESTCONF client connections. Since our security policy only allows authentication for clients having a certificate signed by our CA, we only configure its certificate below.
  </description>
  <trusted-certificate>
    <name>ca.example.com</name>
    <certificate>
      WMdsK2gyTTg3QmtGMjJhWbW1CdFFVAcC30EGrRkYyRTFwdSt4ZVRJbVFFMjLQL1sWp0Cf7Mn6RLR05EMu20VJpK2FWNw2NtDznC4tadVjmZgB8YjkzSfnWSdWwVXCYaYnA4dmtNanFtZj3ma3RgZHXeFppUtbnTWZTF2ZwotNGcEk3UE90cnNFVjJwTUNb0VbQWFPQ0FSSXdn20VPCk1CMEqRMVbUKGd
      VEjJi20JTWEmlbUEKmhpRVHvOTkVvVHFLNw4cfFJB210YUU0CERz05ER
      V6QVJCC05WTQFNvXrT1NUQ0JkYzNOMvPxY0NDUUMRHBNSl16UG8zREF
      NQmdOVkhSTUJZ2jhFCkFqOQFQNRQHFZTVJZER3RU1vdfpQXxJSGdEQnBC
      Z05WSF14Rv1qQmdNRJznSXFBZ2hoNW9kSFJ3I2k4d1pYaGqYljC1pTN
      WpiJMB2W1hoa4GJYQnNaUzVqYr215aU9LTJNNFFQ3pBSkJnT1ZCQV1UQWQmdOVkJBVWRBBf2UTVJBD0rWURUWVFRLxdkbAp1R0Z0Y0d4Be1RNHHEQ
      MKF6a3hQUd1QWHR0dSV1U1eUc1SR0Wm0vK3B0R2F1eXVDMjBRd2kVZ
      25P3npZNEhONApXY0pTa0pZEK2xtYWw3RTRORUZXS9RdGp4NLU1ZXmdvN2
      RJSUQFRRStS0Cg==
    </certificate>
  </trusted-certificate>
</trusted-certificates>
The following example illustrates a "certificate-expiration" notification in XML.

```xml
<notification
 xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2016-07-08T00:01:00Z</eventTime>
 <certificate-expiration
 xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">
 <certificate>
 </certificate>
 <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
 </certificate-expiration>
</notification>
```

4.1.3. YANG Model

This YANG module makes extensive use of data types defined in [RFC5280] and [RFC5958].

```yml
<CODE BEGINS> file "ietf-system-keychain@2016-03-16.yang"

module ietf-system-keychain {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-system-keychain";
  prefix "kc";

  import ietf-yang-types { // RFC 6991
```

Watsen & Schoenwaelder Expires September 17, 2016 [Page 18]
prefix yang;
}

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

contact
"WG Web:  <http://tools.ietf.org/wg/netconf/>
WG List:  <mailto:netconf@ietf.org>

WG Chair: Mehmet Ersue
  <mailto:mehmet.ersue@nsn.com>

WG Chair: Mahesh Jethanandani
  <mailto:mjethanandani@gmail.com>

Editor:  Kent Watsen
  <mailto:kwatsen@juniper.net>";

description
"This module defines a keychain to centralize management of
security credentials.

Copyright (c) 2014 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
(http://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC VVVV; see
the RFC itself for full legal notices."

revision "2016-03-16" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
     Models";
}

typedef algorithms {
  type enumeration {
    enum rsa { description "The RSA algorithm."; } }
enum secp192r1 { description "The secp192r1 algorithm."; }
enum secp256r1 { description "The secp256r1 algorithm."; }
enum secp384r1 { description "The secp384r1 algorithm."; }
enum secp521r1 { description "The secp521r1 algorithm."; }

// what about ecdh_x25519 and ecdh_x448 in TLS 1.3?

// Asymmetric key algorithms. This list has been trimmed down to the minimal subset of algorithms recommended by the IETF. Please see the Design Consideration section in RFC VVVV for more information about this.

container keychain { 
  description "A list of private-keys and their associated certificates, as well as lists of trusted certificates for client certificate authentication. RPCs are provided to generate a new private key and to generate a certificate signing requests.";
}

container private-keys { 
  description "A list of private key maintained by the keychain.";
  list private-key { 
    key name; 
    description "A private key.";
    leaf name { 
      type string; 
      description "An arbitrary name for the private key.";
    }
    leaf algorithm { 
      type kc:algorithms; 
      config false; 
      description "The algorithm used by the private key.";
    }
    leaf key-length { 
      type uint32; 
      config false; 
      description "The key-length used by the private key.";
    }
    leaf public-key { 
      type binary; 
      config false; 
      mandatory true; 
    }
  }
}
description
"An OneAsymmetricKey 'publicKey' structure as specified by RFC 5958, Section 2 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.";
reference
"RFC 5958:
Asymmetric Key Packages
ITU-T X.690:
Information technology - ASN.1 encoding rules:
Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).";
}
container certificate-chains {
  description
  "Certificate chains associated with this private key. More than one chain per key is enabled to support, for instance, a TPM-protected key that has associated both IDevID and LDevID certificates.";
  list certificate-chain {
    key name;
    description
      "A certificate chain for this public key.";
    leaf name {
      type string;
      description
        "An arbitrary name for the certificate chain.";
    }
    leaf-list certificate {
      type binary;
      ordered-by user;
      description
        "An X.509 v3 certificate structure as specified by RFC 5280, Section 4 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690. The list of certificates that run from the server certificate towards the trust anchor. The chain MAY include the trust anchor certificate itself.";
    reference
    "RFC 5280:
    ITU-T X.690:
      Information technology - ASN.1 encoding rules:
      Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).";
  }
}
action generate-certificate-signing-request {
    description
        "Generates a certificate signing request structure for
        the associated private key using the passed subject and
        attribute values. Please review both the Security
        Considerations and Design Considerations sections in
        RFC VVVV for more information regarding this action
        statement.";
    input {
        leaf subject {
            type binary;
            mandatory true;
            description
                "The 'subject' field from the CertificationRequestInfo
                structure as specified by RFC 2986, Section 4.1 encoded
                using the ASN.1 distinguished encoding rules (DER), as
                specified in ITU-T X.690.";
            reference
                "RFC 2986:
                PKCS #10: Certification Request Syntax Specification
                Version 1.7.
                ITU-T X.690:
                Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
                Encoding Rules (DER).";
        }
        leaf attributes {
            type binary;
            description
                "The 'attributes' field from the CertificationRequestInfo
                structure as specified by RFC 2986, Section 4.1 encoded
                using the ASN.1 distinguished encoding rules (DER), as
                specified in ITU-T X.690.";
            reference
                "RFC 2986:
                PKCS #10: Certification Request Syntax Specification
                Version 1.7.
                ITU-T X.690:
                Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
                Encoding Rules (DER).";
        }
    }
}
output {
  leaf certificate-signing-request {
    type binary;
    mandatory true;
    description
      "A CertificationRequest structure as specified by RFC 2986, Section 4.1 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.";
    reference
      "RFC 2986:
      PKCS #10: Certification Request Syntax Specification
      Version 1.7.
      ITU-T X.690:
      Information technology - ASN.1 encoding rules:
      Specification of Basic Encoding Rules (BER),
      Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).";
  }
}
}

action generate-private-key {
  description
    "Requests the device to generate a private key using the specified algorithm and key length.";
  input {
    leaf name {
      type string;
      mandatory true;
      description
        "The name this private-key should have when listed in /keychain/private-keys. As such, the passed value must not match any existing 'name' value.";
    }
    leaf key-usage {
      type enumeration {
        enum signing    { description "signing"; }
        enum encryption { description "encryption"; }
        // unclear if these should be somehow more specific or varied.
      }
      description
        "An optional parameter further restricting the use of this key. Some algorithms inherently restrict use (DH for signing) whereas others can support more than one use (RSA). This flag forces the device to only
allow the key to be used for the indicated purposes.

leaf algorithm {
    type kc:algorithms;
    mandatory true;
    description
        "The algorithm to be used when generating the key.";
}
leaf key-length {
    type uint32;
    description
        "For algorithms that need a key length specified
         when generating the key.";
}
}

action load-private-key {
    description
        "Requests the device to load a private key";
    input {
        leaf name {
            type string;
            mandatory true;
            description
                "The name this private-key should have when listed
                 in /keychain/private-keys. As such, the passed
                 value must not match any existing 'name' value.";
        }
        leaf private-key {
            type binary;
            mandatory true;
            description
                "An OneAsymmetricKey structure as specified by RFC
                5958, Section 2 encoded using the ASN.1 distinguished
                encoding rules (DER), as specified in ITU-T X.690.
                Note that this is the raw private with no shrouding
                to protect it. The strength of this private key
                MUST NOT be greater than the strength of the secure
                connection over which it is communicated. Devices
                SHOULD fail this request if ever that happens.";
                reference
                "RFC 5958: Asymmetric Key Packages
                ITU-T X.690: Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
list trusted-certificates {
  key name;
  description
    "A list of trusted certificates. Each list SHOULD be specific to a purpose. For instance, there could be one list for authenticating NETCONF/RESTCONF client certificates, and another list for authenticating manufacturer-signed data, and yet another list for authenticated web servers.";
  leaf name {
    type string;
    description
      "An arbitrary name for this list of trusted certificates.";
  }
  leaf description {
    type string;
    description
      "An arbitrary description for this list of trusted certificates.";
  }
}
list trusted-certificate {
  key name;
  description
    "A trusted certificate for a specific use.";
  leaf name {
    type string;
    description
      "An arbitrary name for this trusted certificate.";
  }
  leaf certificate {
    type binary;
    description
      "An X.509 v3 certificate structure as specified by RFC 5280, Section 4 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.";
    reference
      "RFC 5280:
      ITU-T X.690:
      Information technology - ASN.1 encoding rules:
      Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)";
  }
4.2. The SSH Server Model

The SSH Server model presented in this section presents two YANG groupings, one for a server that opens a socket to accept TCP connections on, and another for a server that has had the TCP connection opened for it already (e.g., inetd).

The SSH Server model (like the TLS Server model presented below) is provided as a grouping so that it can be used in different contexts. For instance, the NETCONF Server model presented in Section 4.4 uses one grouping to configure a NETCONF server listening for connections and the other grouping to configure NETCONF call home.

A shared characteristic between both groupings is the ability to configure which host key is presented to clients, the private key for which is held in the keychain configuration presented before. Another shared characteristic is the ability to configure which
trusted CA or client certificates the server should be used to authenticate clients when using X.509 based client certificates [RFC6187].

4.2.1. Tree Diagram

The following tree diagram represents the data model for the grouping used to configure an SSH server to listen for TCP connections. The tree diagram for the other grouping is not provided, but it is the same except without the "address" and "port" fields.

NOTE: the diagram below shows "listening-ssh-server" as a YANG container (not a grouping). This temporary container was created only to enable the 'pyang' tool to output the tree diagram, as groupings by themselves have no protocol accessible nodes, and hence 'pyang' would output an empty tree diagram.

module: ietf-ssh-server
  +--rw listening-ssh-server
    +--rw address?            inet:ip-address
    +--rw port                inet:port-number
    +--rw host-keys
        +--rw host-key* [name]
            +--rw name           string
            +--rw (type)?
                +--:(public-key)
                    +--rw public-key?    -> /kc:keychain/private-keys/private-key/name
                +--:(certificate)
    +--rw client-cert-auth {ssh-x509-certs}?
        +--rw trusted-ca-certs?   -> /kc:keychain/trusted-certificates/name
        +--rw trusted-client-certs?   -> /kc:keychain/trusted-certificates/name

4.2.2. Example Usage

This section shows how it would appear if the temporary listening-ssh-server container just mentioned above were populated with some data. This example is consistent with the examples presented earlier in this document.
<listening-ssh-server
   xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">
   <port>830</port>
   <host-keys>
      <host-key>
         <name>deployment-specific-certificate</name>
         <certificate>ex-key-sect571r1-cert</certificate>
      </host-key>
   </host-keys>
   <client-cert-auth>
      <trusted-ca-certs>
         deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
         explicitly-trusted-client-certs
      </trusted-client-certs>
   </client-cert-auth>
</listening-ssh-server>

4.2.3. YANG Model

This YANG module has a normative reference to [RFC4253].

<CODE BEGINS> file "ietf-ssh-server@2016-03-16.yang"

module ietf-ssh-server {
   yang-version 1.1;

   namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
   prefix "ts";

   import ietf-inet-types { // RFC 6991
      prefix inet;
   }
   import ietf-system-keychain { // RFC VVVV
      prefix kc;
      revision-date 2016-03-16;
   }

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

   contact
      "WG Web:   <http://tools.ietf.org/wg/netconf/>
      WG List:  <mailto:netconf@ietf.org>"
description
"This module defines a reusable grouping for a SSH server that can be used as a basis for specific SSH server instances.

Copyright (c) 2014 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 (http://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC VVVV; see the RFC itself for full legal notices."

revision "2016-03-16" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// features
feature ssh-x509-certs {
  description
    "The ssh-x509-certs feature indicates that the NETCONF server supports RFC 6187";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}

// grouping
grouping non-listening-ssh-server-grouping {
  description

"A reusable grouping for a SSH server that can be used as a basis for specific SSH server instances."

container host-keys {
  description
    "The list of host-keys the SSH server will present when establishing a SSH connection.";
  list host-key {
    key name;
    min-elements 1;
    ordered-by user;
    description
      "An ordered list of host keys the SSH server will use to construct its ordered list of algorithms, when sending its SSH_MSG_KEXINIT message, as defined in Section 7.1 of RFC 4253.";
    reference
      "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    leaf name {
      type string;
      mandatory true;
      description
        "An arbitrary name for this host-key";
    }
    choice type {
      description
        "The type of host key being specified";
      leaf public-key {
        type leafref {
          path "/kc:keychain/kc:private-keys/kc:private-key/" + "kc:name";
        }
        description
          "The public key is actually identified by the name of its cooresponding private-key in the keychain.";
      }
      leaf certificate {
        if-feature ssh-x509-certs;
        type leafref {
        }
        description
          "The name of a certificate in the keychain.";
      }
    }
  }
}
container client-cert-auth {
  if-feature ssh-x509-certs;
  description
    "A reference to a list of trusted certificate authority (CA)
    certificates and a reference to a list of trusted client
    certificates.";
  leaf trusted-ca-certs {
    type leafref {
      path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
    description
      "A reference to a list of certificate authority (CA)
      certificates used by the SSH server to authenticate
      SSH client certificates.";
  }
  leaf trusted-client-certs {
    type leafref {
      path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
    description
      "A reference to a list of client certificates used by
      the SSH server to authenticate SSH client certificates.
      A clients certificate is authenticated if it is an
      exact match to a configured trusted client certificate.";
  }
}

grouping listening-ssh-server-grouping {
  description
    "A reusable grouping for a SSH server that can be used as a
    basis for specific SSH server instances.";
  leaf address {
    type inet:ip-address;
    description
      "The IP address of the interface to listen on. The SSH
      server will listen on all interfaces if no value is
      specified.";
  }
  leaf port {
    type inet:port-number;
    mandatory true;  // will a default augmented in work?
    description
      "The local port number on this interface the SSH server

4.3. The TLS Server Model

The TLS Server model presented in this section presents two YANG groupings, one for a server that opens a socket to accept TCP connections on, and another for a server that has had the TCP connection opened for it already (e.g., inetd).

The TLS Server model (like the SSH Server model presented above) is provided as a grouping so that it can be used in different contexts. For instance, the NETCONF Server model presented in Section 4.4 uses one grouping to configure a NETCONF server listening for connections and the other grouping to configure NETCONF call home.

A shared characteristic between both groupings is the ability to configure which server certificate is presented to clients, the private key for which is held in the keychain model presented in Section 4.1. Another shared characteristic is the ability to configure which trusted CA or client certificates the server should be used to authenticate clients.

4.3.1. Tree Diagram

The following tree diagram represents the data model for the grouping used to configure an TLS server to listen for TCP connections. The tree diagram for the other grouping is not provided, but it is the same except without the "address" and "port" fields.
NOTE: the diagram below shows "listening-ssh-server" as a YANG container (not a grouping). This temporary container was created only to enable the "pyang" tool to output the tree diagram, as groupings by themselves have no protocol accessible nodes, and hence "pyang" would output an empty tree diagram.

module: ietf-tls-server
   +--rw listening-tls-server
      +--rw address?           inet:ip-address
      +--rw port               inet:port-number
      +--rw certificates
         +--rw certificate* [name]
            +--rw name    -> /kc:keychain/private-keys/private-key/certificate-chains/certificate-chain/certificate
         +--rw client-auth
         +--rw trusted-ca-certs?    -> /kc:keychain/trusted-certificates/name
         +--rw trusted-client-certs?  -> /kc:keychain/trusted-certificates/name

4.3.2. Example Usage

   <listening-tls-server
      xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
      <port>6513</port>
      <certificates>
         <certificate>
            <name>ex-key-sect571r1-cert</name>
         </certificate>
      </certificates>
      <client-auth>
         <trusted-ca-certs>
            deployment-specific-ca-certs
         </trusted-ca-certs>
         <trusted-client-certs>
            explicitly-trusted-client-certs
         </trusted-client-certs>
      </client-auth>
   </listening-tls-server>

4.3.3. YANG Model

   <CODE BEGINS> file "ietf-tls-server@2016-03-16.yang"

   module ietf-tls-server {
      yang-version 1.1;

   Watsen & Schoenwaelder Expires September 17, 2016 [Page 33]
namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
prefix "ts";

import ietf-inet-types {               // RFC 6991
  prefix inet;
}
import ietf-system-keychain {          // RFC VVVV
  prefix kc;
  revision-date 2016-03-16;
}

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

contact
  "WG Web:  <http://tools.ietf.org/wg/netconf/>
  WG List:  <mailto:netconf@ietf.org>
  WG Chair: Mehmet Ersue
    <mailto:mehmet.ersue@nsn.com>
  WG Chair: Mahesh Jethanandani
    <mailto:mjethanandani@gmail.com>
  Editor:   Kent Watsen
    <mailto:kwatsen@juniper.net>";

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

Copyright (c) 2014 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 (http://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC VVVV; see the RFC itself for full legal notices.";

revision "2016-03-16" {
  description
    "Initial version";
reference
"RFC VVVV: NETCONF Server and RESTCONF Server Configuration
Models";
}

// grouping
grouping non-listening-tls-server-grouping {
  description
  "A reusable grouping for a TLS server that can be used as a
  basis for specific TLS server instances.";
  container certificates {
    description
      "The list of certificates the TLS server will present when
      establishing a TLS connection in its Certificate message,
      as defined in Section 7.4.2 in RRC 5246.";
    reference
    list certificate {
      key name;
      min-elements 1;
      description
        "An unordered list of certificates the TLS server can pick
        from when sending its Server Certificate message.";
      reference
        "RFC 5246: The TLS Protocol, Section 7.4.2";
    leaf name {
      type leafref {
        path "/kc:keychain/kc:private-keys/kc:private-key/
          + "kc:certificate-chains/kc:certificate-chain/
          + "kc:certificate";
      }
      description
        "The name of the certificate in the keychain.";
    }
  }
}

container client-auth {
  description
    "A reference to a list of trusted certificate authority (CA)
    certificates and a reference to a list of trusted client
    certificates.";
  leaf trusted-ca-certs {
    type leafref {
      path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
  }
}
description
 "A reference to a list of certificate authority (CA) certificates used by the TLS server to authenticate TLS client certificates.";
}

leaf trusted-client-certs {
    type leafref {
        path "/kc:keychain/kc:trusted-certificates/kc:name";
    }

description
 "A reference to a list of client certificates used by the TLS server to authenticate TLS client certificates. A client certificate is authenticated if it is an exact match to a configured trusted client certificate.";
}
}

grouping listening-tls-server-grouping {
    description
    "A reusable grouping for a TLS server that can be used as a basis for specific TLS server instances.";

    leaf address {
        type inet:ip-address;
        description
        "The IP address of the interface to listen on. The TLS server will listen on all interfaces if no value is specified.";
    }

    leaf port {
        type inet:port-number;
        mandatory true;  // will a default augmented in work?
        description
        "The local port number on this interface the TL TLS server listens on.";
    }

    uses non-listening-tls-server-grouping;
}

container listening-tls-server {
    description
    "This container will be removed by the RFC Editor. This container is currently only present in order to enable the `pyang` tool to generate tree diagram output of this module (used in the draft) as it otherwise would not contain any protocol accessible nodes to output.";
}
4.4. The NETCONF Server Model

The NETCONF Server model presented in this section supports servers both listening for connections to accept as well as initiating call-home connections. This model also supports both the SSH and TLS transport protocols, using the SSH Server and TLS Server groupings presented in Section 4.2 and Section 4.3 respectively. All private keys and trusted certificates are held in the keychain model presented in Section 4.1. YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

4.4.1. Tree Diagram

The following tree diagram uses line-wrapping in order to comply with xml2rfc validation. This is annoying as I find that drafts (even txt drafts) look just fine with long lines - maybe xml2rfc should remove this warning? - or pyang could have an option to suppress printing leafref paths?

```
module: ietf-netconf-server
  +--rw netconf-server
    +--rw session-options
    |   +--rw hello-timeout? uint16
    +--rw listen {(ssh-listen or tls-listen)}?
      +--rw max-sessions? uint16
      +--rw idle-timeout? uint16
      +--rw endpoint* [name]
        +--rw name string
        +--rw (transport)
          +--:(ssh) {ssh-listen}?
            +--rw address? inet:ip-address
            +--rw port inet:port-number
            +--rw host-keys
              +--rw host-key* [name]
                +--rw name string
                +--rw (type)?
                  +--:(public-key)
                    +--rw public-key? -> /kc:keychain/p
```
rivate-keys/private-key/name
|--|--:(certificate)
|  |--++rw certificate?  -> /kc:keychain/p
private-keys/private-key/certificate-chains/certificate-chain/certificate {ssh-x509-certs}?
|--++rw client-cert-auth {ssh-x509-certs}?
    |--++rw trusted-ca-certs?  -> /kc:keychain/t
trusted-certificates/name
|--++rw trusted-client-certs?  -> /kc:keychain/t
trusted-certificates/name
|--++:(tls) {tls-listen}?
    |--++rw tls
        |--++rw address?  inet:ip-address
        |--++rw port  inet:port-number
        |--++rw certificates
            |--++rw certificate* [name]
                |--++rw name  -> /kc:keychain/private-keys/p
private-key/certificate-chains/certificate-chain/certificate
|--++rw client-auth
    |--++rw trusted-ca-certs?  -> /kc:keychain/t
trusted-certificates/name
|--++rw trusted-client-certs?  -> /kc:keychain/t
trusted-certificates/name
|--++rw cert-maps
    |--++rw cert-to-name* [id]
        |--++rw id  uint32
        |--++rw fingerprint  x509c2n:tls-fingerprint
|--++rw call-home {ssh-call-home or tls-call-home})?
    |--++rw netconf-client* [name]
        |--++rw name  string
        |--++rw (transport)
            |--++:(ssh) {ssh-call-home}?
                |--++rw ssh
                    |--++rw endpoints
                        |--++rw endpoint* [name]
                            |--++rw name  string
                            |--++rw address  inet:host
                            |--++rw port?  inet:port-number
                        |--++rw host-keys
                            |--++rw host-key* [name]
                                |--++rw name  string
                                |--++rw (type)?
                                    |--++:(public-key)
                                        |--++rw public-key?  -> /kc:keychain/p
private-keys/private-key/name
+--:(certificate)
    +--rw client-cert-auth (ssh-x509-certs)?
    +--rw trusted-ca-certs? -> /kc:keychain/tusted-certificates/name
    +--rw trusted-client-certs? -> /kc:keychain/tusted-certificates/name
rusted-certificates/name
  +--:(tls) {tls-call-home}?
    +--rw tls
      +--rw endpoints
        +--rw endpoint* [name]
          +--rw name string
          +--rw address inet:host
          +--rw port? inet:port-number
        +--rw certificates
          +--rw certificate* [name]
            +--rw name -> /kc:keychain/private-keys/private-key/certificate-chains/certificate-chain/certificate
          +--rw client-auth
            +--rw trusted-ca-certs? -> /kc:keychain/tusted-certificates/name
            +--rw trusted-client-certs? -> /kc:keychain/tusted-certificates/name
            +--rw cert-maps
              +--rw cert-to-name* [id]
                +--rw id uint32
                +--rw fingerprint x509c2n:tls-fingerprint
    +--rw connection-type
      +--rw (connection-type)?
        +--rw persistent!
          +--rw idle-timeout? uint32
          +--rw keep-ales
        +--rw max-wait? uint16
          +--rw max-attempts? uint8
      +--:(periodic-connection)
        +--rw periodic!
          +--rw idle-timeout? uint16
          +--rw reconnect_timeout? uint16
        +--rw reconnect-strategy
          +--rw start-with? enumeration
          +--rw max-attempts? uint8
4.4.2. Example Usage

Configuring a NETCONF Server to listen for NETCONF client connections using both the SSH and TLS transport protocols, as well as configuring call-home to two NETCONF clients, one using SSH and the other using TLS.

This example is consistent with other examples presented in this document.

```xml
<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <listen>

    <!-- listening for SSH connections -->
    <endpoint>
      <name>netconf/ssh</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>
            <public-key>my-rsa-key</public-key>
          </host-key>
          <host-key>
            <certificate>TPM key</certificate>
          </host-key>
        </host-keys>
        <client-cert-auth>
          <trusted-ca-certs>
            deployment-specific-ca-certs
          </trusted-ca-certs>
          <trusted-client-certs>
            explicitly-trusted-client-certs
          </trusted-client-certs>
        </client-cert-auth>
      </ssh>
    </endpoint>

    <!-- listening for TLS connections -->
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>ex-key-sect571r1-cert</certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>
          </trusted-ca-certs>
      </tls>
    </endpoint>

```
deployment-specific-ca-certs
</trusted-ca-certs>
<trusted-client-certs>
  explicitly-trusted-client-certs
</trusted-client-certs>
<cert-maps>
  <cert-to-name>
    <id>1</id>
    <fingerprint>11:0A:05:11:00</fingerprint>
    <map-type>x509c2n:san-any</map-type>
  </cert-to-name>
  <cert-to-name>
    <id>2</id>
    <fingerprint>B3:4F:A1:8C:54</fingerprint>
    <map-type>x509c2n:specified</map-type>
    <name>scooby-doo</name>
  </cert-to-name>
</cert-maps>
</client-auth>
</tls>
</endpoint>

</listen>
<call-home>

<!-- calling home to an SSH-based NETCONF client -->
<netconf-client>
  <name>config-mgr</name>
  <ssh>
    <endpoints>
      <endpoint>
        <name>east-data-center</name>
        <address>11.22.33.44</address>
      </endpoint>
      <endpoint>
        <name>west-data-center</name>
        <address>55.66.77.88</address>
      </endpoint>
    </endpoints>
    <host-keys>
      <host-key>
        <certificate>TPM key</certificate>
      </host-key>
    </host-keys>
    <client-cert-auth>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
    </client-cert-auth>
  </ssh>
</netconf-client>

</call-home>
<trusted-client-certs>
  explicitly-trusted-client-certs
</trusted-client-certs>
</client-cert-auth>
</ssh>
<connection-type>
  <periodic>
    <idle-timeout>300</idle-timeout>
    <reconnect-timeout>60</reconnect-timeout>
  </periodic>
</connection-type>
<reconnect-strategy>
  <start-with>last-connected</start-with>
  <max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>

<!-- calling home to a TLS-based NETCONF client -->
<netconf-client>
  <name>event-correlator</name>
  <tls>
    <endpoints>
      <endpoint>
        <name>east-data-center</name>
        <address>22.33.44.55</address>
      </endpoint>
      <endpoint>
        <name>west-data-center</name>
        <address>33.44.55.66</address>
      </endpoint>
    </endpoints>
    <certificates>
      <certificate>ex-key-sect571r1-cert</certificate>
    </certificates>
    <client-auth>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
        explicitly-trusted-client-certs
      </trusted-client-certs>
      <cert-maps>
        <cert-to-name>
          <id>1</id>
          <fingerprint>11:0A:05:11:00</fingerprint>
          <map-type>x509c2n:san-any</map-type>
        </cert-to-name>
      </cert-maps>
    </client-auth>
  </tls>
</netconf-client>
<id>2</id>
<fingerprint>B3:4F:A1:8C:54</fingerprint>
<map-type>x509c2n:specified</map-type>
<name>scooby-doo</name>
</cert-to-name>
</cert-maps>
</client-auth>
</tls>
<connection-type>
<persistent>
<idle-timeout>300</idle-timeout>
<keep-alives>
<max-wait>30</max-wait>
<max-attempts>3</max-attempts>
</keep-alives>
</persistent>
</connection-type>
<reconnect-strategy>
<start-with>first-listed</start-with>
<max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>
</call-home>
</netconf-server>

4.4.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-netconf-server@2016-03-16.yang"

module ietf-netconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
  prefix "ncserver";

  import ietf-inet-types { // RFC 6991
    prefix inet;
  }
  import ietf-x509-cert-to-name { // RFC 7407
    prefix x509c2n;
  }
  import ietf-ssh-server { // RFC VVVV
    prefix ss;
    revision-date 2016-03-16;
  }

</CODE ENDS>
import ietf-tls-server { // RFC VVVV
    prefix ts;
    revision-date 2016-03-16;
}

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

contact
"WG Web: <http://tools.ietf.org/wg/netconf/>
WG List: <mailto:netconf@ietf.org>

WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nsn.com>

WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com>

Editor: Kent Watsen
<mailto:kwatsen@juniper.net">

description
"This module contains a collection of YANG definitions for
configuring NETCONF servers.

Copyright (c) 2014 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
(http://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC VVVV; see
the RFC itself for full legal notices.
"

revision "2016-03-16" {
    description
    "Initial version";
    reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
     Models";
}
// Features

feature ssh-listen {
  description
  "The ssh-listen feature indicates that the NETCONF server
  supports opening a port to accept NETCONF over SSH
  client connections.";
  reference
  "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature ssh-call-home {
  description
  "The ssh-call-home feature indicates that the NETCONF
  server supports initiating a NETCONF over SSH call
  home connection to NETCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
  description
  "The tls-listen feature indicates that the NETCONF server
  supports opening a port to accept NETCONF over TLS
  client connections.";
  reference
  "RFC 7589: Using the NETCONF Protocol over Transport
  Layer Security (TLS) with Mutual X.509 Authentication";
}

feature tls-call-home {
  description
  "The tls-call-home feature indicates that the NETCONF server
  supports initiating a NETCONF over TLS call
  home connection to NETCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature ssh-x509-certs {
  description
  "The ssh-x509-certs feature indicates that the NETCONF server supports RFC 6187";
  reference
  "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}
// top-level container (groupings below)
container netconf-server {
  description
  "Top-level container for NETCONF server configuration."
}

container session-options {
  description
  "NETCONF session options, independent of transport
  or connection strategy."
  leaf hello-timeout {
    type uint16;
    units "seconds";
    default 600;
    description
    "Specifies the maximum number of seconds that a SSH/TLS
    connection may wait for a hello message to be received.
    A connection will be dropped if no hello message is
    received before this number of seconds elapses. If set
to zero, then the server will wait forever for a hello
    message.";
  }
}

container listen {
  if-feature "(ssh-listen or tls-listen)"
  description
  "Configures listen behavior"
  leaf max-sessions {
    type uint16;
    default 0;
    description
    "Specifies the maximum number of concurrent sessions
    that can be active at one time. The value 0 indicates
    that no artificial session limit should be used.";
  }
  leaf idle-timeout {
    type uint16;
    units "seconds";
    default 3600; // one hour
    description
    "Specifies the maximum number of seconds that a NETCONF
    session may remain idle. A NETCONF session will be dropped
    if it is idle for an interval longer than this number of
    seconds. If set to zero, then the server will never drop
    a session because it is idle. Sessions that have a
    notification subscription active are never dropped.";
  }
  list endpoint {

key name;
   description
   "List of endpoints to listen for NETCONF connections on.";
leaf name {
   type string;
   description
   "An arbitrary name for the NETCONF listen endpoint.";
}
choice transport {
   mandatory true;
   description
   "Selects between available transports.";
   case ssh {
      if-feature ssh-listen;
      container ssh {
         description
         "SSH-specific listening configuration for inbound
         connections.";
         uses ss:listening-ssh-server-grouping {
            refine port {
               default 830;
            }
         }
      }
   }
   case tls {
      if-feature tls-listen;
      container tls {
         description
         "TLS-specific listening configuration for inbound
         connections.";
         uses ts:listening-tls-server-grouping {
            refine port {
               default 6513;
            }
            augment "client-auth" {
               description
               "Augments in the cert-to-name structure.";
               uses cert-maps-grouping;
            }
         }
      }
   }
}
container call-home {
if-feature "(ssh-call-home or tls-call-home)";

description
"Configures call-home behavior";

list netconf-client {
  key name;
  description
  "List of NETCONF clients the NETCONF server is to initiate call-home connections to.";

  leaf name {
    type string;
    description
    "An arbitrary name for the remote NETCONF client.";
  }

  choice transport {
    mandatory true;
    description
    "Selects between available transports.";

    case ssh {
      if-feature ssh-call-home;
      container ssh {
        description
        "Specifies SSH-specific call-home transport configuration.";

        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 7777;
          }
        }

        uses ss:non-listening-ssh-server-grouping;
      }
    }

    case tls {
      if-feature tls-call-home;
      container tls {
        description
        "Specifies TLS-specific call-home transport configuration.";

        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 8888;
          }
        }

        uses ts:non-listening-tls-server-grouping {
          augment "client-auth" {
            description
            "Augments in the cert-to-name structure.";
            uses cert-maps-grouping;
          }
        }
      }
    }
  }
}
container connection-type {
  description
  "Indicates the kind of connection to use.";
  choice connection-type {
    description
    "Selects between available connection types.";
    case persistent-connection {
      container persistent {
        presence true;
        description
        "Maintain a persistent connection to the NETCONF
        client. If the connection goes down, immediately
        start trying to reconnect to it, using the
        reconnection strategy.

        This connection type minimizes any NETCONF client
to NETCONF server data-transfer delay, albeit at
the expense of holding resources longer."
      };
      leaf idle-timeout {
        type uint32;
        units "seconds";
        default 86400;  // one day;
        description
        "Specifies the maximum number of seconds that a
        NETCONF session may remain idle. A NETCONF
        session will be dropped if it is idle for an
        interval longer than this number of seconds.
        If set to zero, then the server will never drop
        a session because it is idle. Sessions that
        have a notification subscription active are
        never dropped."
      }
    }
  }
  container keep-alives {
    description
    "Configures the keep-alive policy, to proactively
    test the aliveness of the SSH/TLS client. An
    unresponsive SSH/TLS client will be dropped after
    approximately max-attempts * max-wait seconds.";
    reference
    "RFC YYYY: NETCONF Call Home and RESTCONF Call
    Home, Section 3.1, item S6";
    leaf max-wait {
      type uint16 {
        range "1..max";
leaf idle-timeout {
    type uint16;
    units "seconds";
    default 300; // five minutes
    description "Specifies the maximum number of seconds that a NETCONF session may remain idle. A NETCONF session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
}
leaf reconnect_timeout {
    type uint16 {
        range "1..max";
    }
units minutes;
default 60;
description
"Sets the maximum amount of unconnected time the
NETCONF server will wait before re-establishing
a connection to the NETCONF client. The NETCONF
server may initiate a connection before this
time if desired (e.g., to deliver an event
notification message).";}
}
}
}
}
}
}
container reconnect-strategy {
  description
  "The reconnection strategy guides how a NETCONF server
reconnects to a NETCONF client, after discovering its
connection to the client has dropped. The NETCONF
server starts with the specified endpoint and tries
to connect to it max-attempts times before trying the
next endpoint in the list (round robin).";
leaf start-with {
  type enumeration {
    enum first-listed {
      description
      "Indicates that reconnections should start with
the first endpoint listed.";
    }
    enum last-connected {
      description
      "Indicates that reconnections should start with
the endpoint last connected to. If no previous
connection has ever been established, then the
first endpoint configured is used. NETCONF
servers SHOULD be able to remember the last
endpoint connected to across reboots.";
    }
  }
  default first-listed;
  description
  "Specifies which of the NETCONF client’s endpoints the
NETCONF server should start with when trying to connect
to the NETCONF client.";
}
leaf max-attempts {
  type uint8 {

range "1..max";
}
default 3;
description
"Specifies the number times the NETCONF server tries to
connect to a specific endpoint before moving on to the
next endpoint in the list (round robin).";
}
}
}
}

grouping cert-maps-grouping {

description
"A grouping that defines a container around the
cert-to-name structure defined in RFC 7407.";

container cert-maps {

grouping endpoints-container {

description
"This grouping is used by both the ssh and tls containers
for call-home configurations.";

container endpoints {

description
"Container for the list of endpoints.";

list endpoint {

leaf name {

}
type string;
description
   "An arbitrary name for this endpoint."
;
leaf address {
   type inet:host;
   mandatory true;
   description
   "The IP address or hostname of the endpoint. If a
   hostname is configured and the DNS resolution results
   in more than one IP address, the NETCONF server
   will process the IP addresses as if they had been
   explicitly configured in place of the hostname."
;
leaf port {
   type inet:port-number;
   description
   "The IP port for this endpoint. The NETCONF server will
   use the IANA-assigned well-known port if no value is
   specified."
;
}

4.5. The RESTCONF Server Model

The RESTCONF Server model presented in this section supports servers
both listening for connections to accept as well as initiating call-
home connections. This model supports the TLS transport only, as
RESTCONF only supports HTTPS, using the TLS Server groupings
presented in Section 4.3. All private keys and trusted certificates
are held in the keychain model presented in Section 4.1. YANG
feature statements are used to enable implementations to advertise
which parts of the model the RESTCONF server supports.

4.5.1. Tree Diagram

The following tree diagram uses line-wrapping in order to comply with
xml2rfc validation. This is annoying as I find that drafts (even txt
drafts) look just fine with long lines - maybe xml2rfc should remove
this warning? - or pyang could have an option to suppress printing
leafref paths?
module: ietf-restconf-server
  +--rw restconf-server
    |  +--rw listen {tls-listen}?
    |     |  +--rw max-sessions? uint16
    |     |  +--rw endpoint* [name]
    |     |     +--rw name string
    |     |     +--rw (transport)
    |     |     +--:(tls) {tls-listen}?
    |     |     |  +--rw tls
    |     |     |     +--rw address? inet:ip-address
    |     |     |     +--rw port inet:port-number
    |     |     |     +--rw certificates
    |     |     |     |  +--rw certificate* [name]
    |     |     |     |     +--rw name -> /kc:keychain/private-keys/private-key/certificate-chains/certificate-chain/certificate
    |     |     |     |     +--rw client-auth
    |     |     |     |     |  +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
    |     |     |     |     +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
    |     |     |     |     +--rw cert-maps
    |     |     |     |     |  +--rw cert-to-name* [id]
    |     |     |     |     |     +--rw id uint32
    |     |     |     |     |     +--rw fingerprint x509c2n:tls-fingerprint
    |     |     |     |     +--rw call-home {tls-call-home}?
    |     |     |     |     +--rw restconf-client* [name]
    |     |     |     |     |  +--rw name string
    |     |     |     |     |     +--rw (transport)
    |     |     |     |     |     +--:(tls) {tls-call-home}?
    |     |     |     |     |     |  +--rw tls
    |     |     |     |     |     |     +--rw endpoints
    |     |     |     |     |     |     |  +--rw endpoint* [name]
    |     |     |     |     |     |     |     +--rw name string
    |     |     |     |     |     |     |     +--rw address inet:host
    |     |     |     |     |     |     |     +--rw port? inet:port-number
    |     |     |     |     |     |     +--rw certificates
    |     |     |     |     |     |     |  +--rw certificate* [name]
    |     |     |     |     |     |     |     +--rw name -> /kc:keychain/private-keys/private-key/certificate-chains/certificate-chain/certificate
    |     |     |     |     |     |     +--rw client-auth
    |     |     |     |     |     |     |  +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
    |     |     |     |     |     |     +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
    |     |     |     |     |     |     +--rw cert-maps
    |     |     |     |     |     |     |  +--rw cert-to-name* [id]
    |     |     |     |     |     |     |     +--rw id uint32
    |     |     |     |     |     |     |     +--rw fingerprint x509c2n:tls-fingerprint
    |     |     |     |     |     |     +--rw call-home {tls-call-home}?
    |     |     |     |     |     +--rw restconf-client* [name]
    |     |     |     |     |     |  +--rw name string
    |     |     |     |     |     |     +--rw (transport)
    |     |     |     |     |     |     +--:(tls) {tls-call-home}?
    |     |     |     |     |     |     |  +--rw tls
    |     |     |     |     |     |     |     +--rw endpoints
    |     |     |     |     |     |     |     |  +--rw endpoint* [name]
    |     |     |     |     |     |     |     |     +--rw name string
    |     |     |     |     |     |     |     |     +--rw address inet:host
    |     |     |     |     |     |     |     |     +--rw port? inet:port-number
    |     |     |     |     |     |     |     +--rw certificates
    |     |     |     |     |     |     |     |  +--rw certificate* [name]
    |     |     |     |     |     |     |     |     +--rw name -> /kc:keychain/private-keys/private-key/certificate-chains/certificate-chain/certificate
    |     |     |     |     |     |     |     +--rw client-auth
    |     |     |     |     |     |     |     |  +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
    |     |     |     |     |     |     |     +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
    |     |     |     |     |     |     |     +--rw cert-maps
4.5.2. Example Usage

Configuring a RESTCONF Server to listen for RESTCONF client connections, as well as configuring call-home to one RESTCONF client.

This example is consistent with other examples presented in this document.

```xml
<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">
  <!-- listening for TLS (HTTPS) connections -->
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>ex-key-sect571r1-cert</certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>
            deployment-specific-ca-certs
          </trusted-ca-certs>
          <trusted-client-certs>
            explicitly-trusted-client-certs
          </trusted-client-certs>
          <cert-maps>
```

Watsen & Schoenwaelder Expires September 17, 2016 [Page 55]
<!-- calling home to a RESTCONF client -->
<call-home>
  <restconf-client>
    <name>config-manager</name>
    <tls>
      <endpoints>
        <endpoint>
          <name>east-data-center</name>
          <address>22.33.44.55</address>
        </endpoint>
        <endpoint>
          <name>west-data-center</name>
          <address>33.44.55.66</address>
        </endpoint>
      </endpoints>
      <certificates>
        <certificate>ex-key-sect571r1-cert</certificate>
      </certificates>
      <client-auth>
        <trusted-ca-certs>
          deployment-specific-ca-certs
        </trusted-ca-certs>
        <trusted-client-certs>
          explicitly-trusted-client-certs
        </trusted-client-certs>
        <cert-maps>
          <cert-to-name>
            <id>1</id>
            <fingerprint>11:0A:05:11:00</fingerprint>
            <map-type>x509c2n:san-any</map-type>
          </cert-to-name>
        </cert-maps>
      </client-auth>
    </tls>
  </restconf-client>
</call-home>
4.5.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-restconf-server@2016-03-16.yang"

module ietf-restconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
  prefix "rcserver";

  //import ietf-netconf-acm {
  //  prefix nacm;                     // RFC 6536
  //}
  import ietf-inet-types {           // RFC 6991
    prefix inet;
  }
  import ietf-x509-cert-to-name {   // RFC 7407
    prefix x509c2n;
  }
  import ietf-tls-server {          // RFC VVVV

  }

<CODE ENDS>
prefix ts;
revision-date 2016-03-16;
}

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

contact
"WG Web: <http://tools.ietf.org/wg/netconf/>
WG List: <mailto:netconf@ietf.org>

WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nsn.com>

WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com>

Editor: Kent Watsen
<mailto:kwatsen@juniper.net>";

description
"This module contains a collection of YANG definitions for configuring RESTCONF servers.

Copyright (c) 2014 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 (http://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC VVVV; see the RFC itself for full legal notices.";

revision "2016-03-16" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features
feature tls-listen {
    description
    "The listen feature indicates that the RESTCONF server
    supports opening a port to listen for incoming RESTCONF
    client connections.";
    reference
    "RFC XXXX: RESTCONF Protocol";
}

feature tls-call-home {
    description
    "The call-home feature indicates that the RESTCONF server
    supports initiating connections to RESTCONF clients.";
    reference
    "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature client-cert-auth {
    description
    "The client-cert-auth feature indicates that the RESTCONF server
    supports the ClientCertificate authentication scheme.";
    reference
    "RFC ZZZZ: Client Authentication over New TLS Connection";
}

// top-level container
container restconf-server {
    description
    "Top-level container for RESTCONF server configuration.";

    container listen {
        if-feature tls-listen;
        description
        "Configures listen behavior";
        leaf max-sessions {
            type uint16;
            default 0;  // should this be 'max'?
            description
            "Specifies the maximum number of concurrent sessions
            that can be active at one time. The value 0 indicates
            that no artificial session limit should be used.";
        }

        list endpoint {
            key name;
            description
            "List of endpoints to listen for RESTCONF connections on.";
            leaf name {

        Watsen & Schoenwaelder Expires September 17, 2016 [Page 59]
type string;
description
   "An arbitrary name for the RESTCONF listen endpoint.";
}
choice transport {
   mandatory true;
description
   "Selects between available transports.";
case tls {
   if-feature tls-listen;
   container tls {
      description
         "TLS-specific listening configuration for inbound
         connections.";
      uses ts:listening-tls-server-grouping {
         refine port {
            default 443;
         }
      }
      augment "client-auth" {
         description
            "Augments in the cert-to-name structure.";
         uses cert-maps-grouping;
      }
   }
}
}
}

container call-home {
   if-feature tls-call-home;
description
   "Configures call-home behavior";
list restconf-client {
   key name;
description
   "List of RESTCONF clients the RESTCONF server is to
   initiate call-home connections to.";
leaf name {
   type string;
description
      "An arbitrary name for the remote RESTCONF client.";
}
choice transport {
   mandatory true;
description
      "Selects between TLS and any transports augmented in.";
}
case tls {
    if-feature tls-call-home;
    container tls {
        description
            "Specifies TLS-specific call-home transport
            configuration.";
        uses endpoints-container {
            refine endpoints/endpoint/port {
                default 9999;
            }
        }
        uses ts:non-listening-tls-server-grouping {
            augment "client-auth" {
                description
                    "Augments in the cert-to-name structure.";
                uses cert-maps-grouping;
            }
        }
    }
    container connection-type {
        description
            "Indicates the RESTCONF client’s preference for how the
            RESTCONF server’s connection is maintained.";
        choice connection-type {
            description
                "Selects between available connection types.";
            case persistent-connection {
                container persistent {
                    presence true;
                    description
                        "Maintain a persistent connection to the RESTCONF
                        client. If the connection goes down, immediately
                        start trying to reconnect to it, using the
                        reconnection strategy.

                        This connection type minimizes any RESTCONF client
                        to RESTCONF server data-transfer delay, albeit at
                        the expense of holding resources longer.";
                }
                container keep-alives {
                    description
                        "Configures the keep-alive policy, to proactively
                        test the aliveness of the TLS client. An
                        unresponsive TLS client will be dropped after
                        approximately (max-attempts * max-wait) seconds.";
                    reference
                }
            }
        }
    }
}
"RFC YYY: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6";

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 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.";
}

case periodic-connection {
  container periodic {
    presence true;
    description
    "Periodically connect to the RESTCONF client, so that
    the RESTCONF client may deliver messages pending for
    the RESTCONF server. The RESTCONF client is expected
to close the connection when it is ready to release
it, thus starting the RESTCONF server’s timer until
next connection.";
    leaf reconnect-timeout {
      type uint16 {
        range "1..max";
      }
      units minutes;
      default 60;
      description
      "The maximum amount of unconnected time the RESTCONF
server will wait before re-establishing a connection
to the RESTCONF client. The RESTCONF server may
initiate a connection before this time if desired
(e.g., to deliver a notification).";
  }
}
container reconnect-strategy {
  description
  "The reconnection strategy guides how a RESTCONF server
  reconnects to an RESTCONF client, after losing a connection
to it, even if due to a reboot. The RESTCONF server starts
with the specified endpoint and tries to connect to it
max-attempts times before trying the next endpoint in the
list (round robin).";

  leaf start-with {
    type enumeration {
      enum first-listed {
        description
        "Indicates that reconnections should start with
        the first endpoint listed.";
      }
      enum last-connected {
        description
        "Indicates that reconnections should start with
        the endpoint last connected to. If no previous
        connection has ever been established, then the
        first endpoint configured is used. RESTCONF
        servers SHOULD be able to remember the last
        endpoint connected to across reboots.";
      }
    }
    default first-listed;
    description
    "Specifies which of the RESTCONF client’s endpoints the
    RESTCONF server should start with when trying to connect
to the RESTCONF client.";
  }

  leaf max-attempts {
    type uint8 {
      range "1..max";
    }
    default 3;
    description
    "Specifies the number times the RESTCONF server tries to
    connect to a specific endpoint before moving on to the
    next endpoint in the list (round robin).";
  }
}
grouping cert-maps-grouping {
    description "A grouping that defines a container around the
cert-to-name structure defined in RFC 7407.";
    container cert-maps {
        uses x509c2n:cert-to-name;
        description "The cert-maps container is used by a TLS-based RESTCONF
server to map the RESTCONF client’s presented X.509
certificate to a RESTCONF username. If no matching and
valid cert-to-name list entry can be found, then the
RESTCONF server MUST close the connection, and MUST NOT
accept RESTCONF messages over it.";
        reference "RFC XXXX: The RESTCONF Protocol";
    }
}

grouping endpoints-container {
    description "This grouping is used by tls container for call-home
configurations.";
    container endpoints {
        description "Container for the list of endpoints.";
        list endpoint {
            key name;
            min-elements 1;
            ordered-by user;
            description "User-ordered list of endpoints for this RESTCONF client.
Defining more than one enables high-availability.";
            leaf name {
                type string;
                description "An arbitrary name for this endpoint.";
            }
            leaf address {
                type inet:host;
                mandatory true;
                description "The IP address or hostname of the endpoint. If a
hostname is configured and the DNS resolution results
in more than one IP address, the RESTCONF server will process the IP addresses as if they had been explicitly configured in place of the hostname."

```yaml
leaf port {
  type inet:port-number;
  description
  "The IP port for this endpoint. The RESTCONF server will use the IANA-assigned well-known port if no value is specified.";
}
```

5. Design Considerations

The manner that the both local and remote endpoints have been specified in the ietf-netconf-server and ietf-rest-server modules does not directly support virtual routing and forwarding (VRF), though they have been specified in such a way to enable external modules will augment in VRF designations when needed.

This document uses PKCS #10 [RFC2986] for the "generate-certificate-signing-request" action. The use of Certificate Request Message Format (CRMF) [RFC4211] was considered, but is was unclear if there was market demand for it, and so support for CRMF has been left out of this specification. If it is desired to support CRMF in the future, placing a "choice" statement in both the input and output statements, along with an "if-feature" statement on the CRMF option, would enable a backwards compatible solution.

This document puts a limit of the number of elliptical curves supported. This was done to match industry trends in IETF best practice (e.g., matching work being done in TLS 1.3). In additional algorithms are needed, they MAY be augmented in by another module, or added directly in a future version of this document.

Both this document and Key Chain YANG Data Model [draft-ietf-rtgwg-yang-key-chain] define keychain YANG modules. The authors looked at this and agree that they two modules server different purposes and hence not worth merging into one document. To
underscore this further, this document renamed its module from "ietf-keychain" to "ietf-system-keychain" and that other document renamed its module from "ietf-key-chain" to "ietf-routing-key-chain".

For the trusted-certificates list, Trust Anchor Format [RFC5914] was evaluated and deemed inappropriate due to this document's need to also support pinning. That is, pinning a client-certificate to support NETCONF over TLS client authentication.

6. Security Considerations

This document defines a keychain mechanism that is entrusted with the safe keeping of private keys, and the safe keeping of trusted certificates. Nowhere in this API is there an ability to access (read out) a private key once it is known to the keychain. Further, associated public keys and attributes (e.g., algorithm name, key length, etc.) are read-only. That said, this document allows for the deletion of private keys and their certificates, as well the deletion of trusted certificates. Access control mechanisms (e.g., NACM [RFC6536]) MUST be in place so as to authorize such client actions. Further, whilst the data model allows for private keys and trusted certificates in general to be deleted, implementations should be well aware that some private keys (e.g., those in a TPM) and some trusted certificates, should never be deleted, regardless if the authorization mechanisms would generally allow for such actions.

For the "generate-certificate-signing-request" action, it is RECOMMENDED that devices implement assert channel binding [RFC5056], so as to ensure that the application layer that sent the request is the same as the device authenticated in the secure transport layer was established.

This document defines a data model that includes a list of private keys. These private keys MAY be deleted using standard NETCONF or RESTCONF operations (e.g., <edit-config>). Implementations SHOULD automatically (without explicit request) zeroize these keys in the most secure manner available, so as to prevent the remnants of their persisted storage locations from being analyzed in any meaningful way.

The keychain module define within this document defines the "load-private-key" action enabling a device to load a client-supplied private key. This is a private key with no shrouding to protect it. The strength of this private key MUST NOT be greater than the strength of the underlying secure transport connection over which it is communicated. Devices SHOULD fail this request if ever the strength of the private key is greater then the strength of the underlying transport.
A denial of service (DoS) attack MAY occur if the NETCONF server limits the maximum number of NETCONF sessions it will accept (i.e. the 'max-sessions' field in the ietf-netconf-server module is not zero) and either the "hello-timeout" or "idle-timeout" fields in ietf-netconf-server module have been set to indicate the NETCONF server should wait forever (i.e. set to zero).

7. IANA Considerations

7.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registrations are requested:

- **URI**: urn:ietf:params:xml:ns:yang:ietf-netconf-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-restconf-server
  - 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 five YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:
8. 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, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, Sean Turner, and Bert Wijnen.

Juergen Schoenwaelder and was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

9. References

9.1. Normative References

[draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home",
draft-ietf-netconf-call-home-02 (work in progress), 2014.
[draft-ietf-netconf-restconf]


9.2. Informative References


Appendix A. Change Log

A.1. 00 to 01
   - Restructured document so it flows better
   - Added trusted-ca-certs and trusted-client-certs objects into the
     ietf-system-tls-auth module

A.2. 01 to 02
   - Removed the "one-to-many" construct
   - Removed "address" as a key field
   - Removed "network-manager" terminology
   - Moved open issues to github issues
   - Brought TLS client auth back into model

A.3. 02 to 03
   - Fixed tree diagrams and surrounding text

A.4. 03 to 04
   - Reduced the number of grouping statements
   - Removed psk-maps and associated feature statements
   - Added ability for listen/call-home instances to specify which
     host-keys/certificates (of all listed) to use
   - Clarified that last-connected should span reboots
   - Added missing "objectives" for selecting which keys to use,
     authenticating client-certificates, and mapping authenticated
     client-certificates to usernames
   - Clarified indirect client certificate authentication
   - Added keep-alive configuration for listen connections
   - Added global-level NETCONF session parameters
A.5. 04 to 05

- Removed all refs to the old ietf-system-tls-auth module
- Removed YANG 1.1 style if-feature statements (loss some expressiveness)
- Removed the read-only (config false) lists of SSH host-keys and TLS certs
- Added an if-feature around session-options container
- Added ability to configure trust-anchors for SSH X.509 client certs
- Now imports by revision, per best practice
- Added support for RESTCONF server
- Added RFC Editor instructions

A.6. 05 to 06

- Removed feature statement on the session-options container (issue #21).
- Added NACM statements to YANG modules for sensitive nodes (issue #24).
- Fixed default RESTCONF server port value to be 443 (issue #26).
- Added client-cert-auth subtree to ietf-restconf-server module (issue #27).
- Updated draft-ietf-netmod-snmp-cfg reference to RFC 7407 (issue #28).
- Added description statements for groupings (issue #29).
- Added description for braces to tree diagram section (issue #30).
- Renamed feature from "rfc6187" to "ssh-x509-certs" (issue #31).

A.7. 06 to 07

- Replaced "application" with "NETCONF/RESTCONF client" (issue #32).
- Reverted back to YANG 1.1 if-feature statements (issue #34).
o Removed import by revisions (issue #36).

o Removed groupings only used once (issue #37).

o Removed upper-bound on hello-timeout, idle-timeout, and max-sessions (issue #38).

o Clarified that when no listen address is configured, the NETCONF/RESTCONF server will listen on all addresses (issue #41).

o Update keep-alive reference to new section in Call Home draft (issue #42).

o Modified connection-type/persistent/keep-alives/interval-secs default value, removed the connection-type/periodic/linger-secs node, and also removed the reconnect-strategy/interval-secs node (issue #43).

o Clarified how last-connected reconnection type should work across reboots (issue #44).

o Clarified how DNS-expanded hostnames should be processed (issue #45).

o Removed text on how to implement keep-alives (now in the call-home draft) and removed the keep-alive configuration for listen connections (issue #46).

o Clarified text for .../periodic-connection/timeout-mins (issue #47).

o Fixed description on the "trusted-ca-certs" leaf-list (issue #48).

o Added optional keychain-based solution in Appendix A (issue #49).

o Fixed description text for the interval-secs leaf (issue #50).

o moved idle-time into the listen, persistent, and periodic subtrees (issue #51).

o put presence statements on containers where it makes sense (issue #53).

A.8. 07 to 08

o Per WG consensus, replaced body with the keychain-based approach described in -07’s Appendix.
A.9.  08 to 09

- Added a lot of introductory text, improved examples, and what not.

- Renamed ietf-keychain to ietf-system-keychain to disambiguate from
  the routing area working group’s keychain model (they similarly
  renamed their model from ietf-key-chain to ietf-routing-key-
  chain).

- Added an action statement to ietf-system-keychain to load a
  private key.

- Added a notification statement to ietf-system-keychain to notify
  when a certificate is nearing expiration and beyond.

- Converted all binary types to use ASN.1 DER encoding.

- Added a Design Considerations section.

- Filled in the Security Considerations section.

- Removed the Other Considerations section.

- Extended the Editorial Note section.

- Added many Normative and Informative references.

Appendix B. Open Issues

Please see: https://github.com/netconf-wg/server-model/issues.

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de