Network Working Group A. Lindem, Ed.
Internet-Draft Y. Qu
Intended status: Standards Track D. Yeung
Expires: April 17, 2016 Cisco Systems
I. Chen
Ericsson
J. Zhang
Juniper Networks
Y. Yang
Cisco Systems
October 15, 2015

Key Chain YANG Data Model
draft-acee-rtg-yang-key-chain-09.txt

Abstract

This document describes the key chain YANG data model. A key chain is a list of elements each containing a key, send lifetime, accept lifetime, and algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, keys and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated. Key chains are commonly used for routing protocol authentication and other applications. In some applications, the protocols do not use the key chain element key directly, but rather a key derivation function is used to derive a short-lived key from the key chain element key.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes the key chain YANG data model. A key chain is a list of elements each containing a key, send lifetime, accept lifetime, and algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, keys and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated. Key chains are commonly used for routing protocol authentication and other applications. In some applications, the protocols do not use the key chain element key directly, but rather a key derivation function is used to derive a short-lived key from the key chain element key.

1.1. Requirements Notation

The key words "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-KEYWORDS].

2. Problem Statement

This document describes a YANG [YANG] data model for key chains. Key chains have been implemented and deployed by a large percentage of network equipment vendors. Providing a standard YANG model will facilitate automated key distribution and non-disruptive key rollover. This will aid in tightening the security of the core routing infrastructure as recommended in [IAB-REPORT].

A key chain is a list containing one or more elements containing a Key ID, key, send/accept lifetimes, and the associated authentication or encryption algorithm. A key chain can be used by any service or application requiring authentication or encryption. In essence, the key-chain is a reusable key policy that can be referenced where ever it is required. The key-chain construct has been implemented by most networking vendors and deployed in many networks.

A conceptual representation of a crypto key table is described in [CRYPTO-KEYTABLE]. The crypto key table also includes keys as well as their corresponding lifetimes and algorithms. Additionally, the key table includes key selection criteria and envisions a deployment model where the details of the applications or services requiring authentication or encryption permeate into the key database. The YANG key-chain model described herein doesn't include key selection criteria or support this deployment model. At the same time, it does not preclude it. The draft [YANG-CRYPTO-KEYTABLE] describes augmentations to the key chain YANG model in support of key selection criteria.

2.1. Graceful Key Rollover using Key Chains

Key chains may be used to gracefully update the key and/or algorithm used by an application for authentication or encryption. This MAY be accomplished by accepting all the keys that have a valid accept lifetime and sending the key with the most recent send lifetime. One scenario for facilitating key rollover is to:

1. Distribute a key chain with a new key to all the routers or other network devices in the domain of that key chain. The new key's accept lifetime should be such that it is accepted during the key rollover period. The send lifetime should be a time in the future when it can be assured that all the routers in the domain

of that key are upgraded. This will have no immediate impact on the keys used for transmission.

2. Assure that all the network devices have been updated with the updated key chain and that their system times are roughly synchronized. The system times of devices within an administrative domain are commonly synchronized (e.g., using Network Time Protocol (NTP) [NTP-PROTO]). This also may be automated.

3. When the send lifetime of the new key becomes valid, the network devices within the domain of key chain will start sending the new key.

4. At some point in the future, a new key chain with the old key removed may be distributed to the network devices within the domain of the key chain. However, this may be deferred until the next key rollover. If this is done, the key chain will always include two keys; either the current and future key (during key rollovers) or the current and previous keys (between key rollovers).

3. Design of the Key Chain Model

The ietf-keychain module contains a list of one or more keys indexed by a Key ID. For some applications (e.g., OSPFv3 [OSPFV3-AUTH]), the Key-Id is used to identify the key chain entry to be used. In addition to the Key-ID, each key chain entry includes a key-string and a cryptographic algorithm. Optionally, the key chain entries include send/accept lifetimes. If the send/accept lifetime is unspecified, the key is always considered valid.

Note that asymmetric keys, i.e., a different key value used for transmission versus acceptance, may be supported with multiple key chain elements where the accept-lifetime or send-lifetime is not valid (e.g., has an end-time equal to the start-time).

Due to the differences in key chain implementations across various vendors, some of the data elements are optional. Additionally, the key-chain is made a grouping so that an implementation could support scoping other than at the global level. Finally, the crypto-algorithm-types grouping is provided for reuse when configuring legacy authentication and encryption not using key-chains.

A key-chain is identified by a unique name within the scope of the network device. The "key-chain-ref" typedef SHOULD be used by other YANG modules when they need to reference a configured key-chain.

3.1. Key Chain Operational State

The key chain operational state is maintained in the key-chain entries along with the configuration state. The key string itself is omitted from the operational state to minimize visibility similar to what was done with keys in SNMP MIBs. This is an area for further discussion. Additionally, the operational state includes an indication of whether or not a key chain entry is valid for sending or acceptance.

3.2. Key Chain Model Features

Features are used to handle differences between vendor implementations. For example, not all vendors support configuration an acceptance tolerance or configuration of key strings in hexadecimal. They are also used to support of security requirements (e.g., TCP-AO Algorithms [TCP-AO-ALGORITHMS]) not implemented by vendors or only a single vendor.

3.3. Key Chain Model Tree

```
+--rw key-chains
   +--rw key-chain-list* [name]
   |  +--rw name                       string
   |  +--ro name-state?                string
   |  +--rw accept-tolerance {accept-tolerance}?
   |  |  +--rw duration?   uint32
   |  +--ro accept-tolerance-state
   |  |  +--ro duration?   uint32
   |  +--rw key-chain-entry* [key-id]
   |     +--rw key-id                       uint64
   |     +--ro key-id-state?                uint64
   |     +--rw key-string
   |     |  +--rw (key-string-style)?
   |     |     +--:(keystring)
   |     |     |  +--rw keystring?            string
   |     |     +--:(hexadecimal) {hex-key-string}?
   |     |        +--rw hexadecimal-string?   yang:hex-string
   |     +--rw lifetime
   |     |  +--rw (lifetime)?
   |     |     +--:(send-and-accept-lifetime)
   |     |     |  +--rw send-accept-lifetime
   |     |     |     +--rw (lifetime)?
   |     |     |        +--:(always)
   |     |     |        |  +--rw always?            empty
   |     |     |        +--:(start-end-time)
   |     |     |           +--rw start-date-time?
   |     |     |                   yang:date-and-time
```

```
     |     |     |           +--rw (end-time)?
     |     |     |              +--:(infinite)
     |     |     |              |  +--rw no-end-time?       empty
     |     |     |              +--:(duration)
     |     |     |              |  +--rw duration?          uint32
     |     |     |              +--:(end-date-time)
     |     |     |                 +--rw end-date-time?
     |     |     |                       yang:date-and-time
     |     |     +--:(independent-send-accept-lifetime)
     |     |             {independent-send-accept-lifetime}?
     |     |        +--rw send-lifetime
     |     |        |  +--rw (lifetime)?
     |     |        |     +--:(always)
     |     |        |     |  +--rw always?            empty
     |     |        |     +--:(start-end-time)
     |     |        |        +--rw start-date-time?
     |     |        |              yang:date-and-time
     |     |        |        +--rw (end-time)?
     |     |        |           +--:(infinite)
     |     |        |           |  +--rw no-end-time?       empty
     |     |        |           +--:(duration)
     |     |        |           |  +--rw duration?          uint32
     |     |        |           +--:(end-date-time)
     |     |        |              +--rw end-date-time?
     |     |        |                    yang:date-and-time
     |     |        +--rw accept-lifetime
     |     |           +--rw (lifetime)?
     |     |              +--:(always)
     |     |              |  +--rw always?            empty
     |     |              +--:(start-end-time)
     |     |                 +--rw start-date-time?
     |     |                 |       yang:date-and-time
     |     |                 +--rw (end-time)?
     |     |                    +--:(infinite)
     |     |                    |  +--rw no-end-time?       empty
     |     |                    +--:(duration)
     |     |                    |  +--rw duration?          uint32
     |     |                    +--:(end-date-time)
     |     |                       +--rw end-date-time?
     |     |                             yang:date-and-time
     |     +--ro lifetime-state
     |     |  +--ro send-lifetime
     |     |  |  +--ro (lifetime)?
     |     |  |     +--:(always)
     |     |  |     |  +--ro always?            empty
     |     |  |     +--:(start-end-time)
     |     |  |        +--ro start-date-time?   yang:date-and-time
     |     |  |        +--ro (end-time)?
```

```
      |     |  |           +--:(infinite)
      |     |  |           |  +--ro no-end-time?       empty
      |     |  |           +--:(duration)
      |     |  |           |  +--ro duration?          uint32
      |     |  |           +--:(end-date-time)
      |     |  |              +--ro end-date-time?
      |     |  |                    yang:date-and-time
      |     |  +--ro send-valid?        boolean
      |     |  +--ro accept-lifetime
      |     |  |  +--ro (lifetime)?
      |     |  |     +--:(always)
      |     |  |     |  +--ro always?            empty
      |     |  |     +--:(start-end-time)
      |     |  |        +--ro start-date-time?   yang:date-and-time
      |     |  |        +--ro (end-time)?
      |     |  |           +--:(infinite)
      |     |  |           |  +--ro no-end-time?       empty
      |     |  |           +--:(duration)
      |     |  |           |  +--ro duration?          uint32
      |     |  |           +--:(end-date-time)
      |     |  |              +--ro end-date-time?
      |     |  |                      yang:date-and-time
      |     |  +--ro accept-valid?      boolean
      |     +--rw crypto-algorithm
      |     |  +--rw (algorithm)?
      |     |     +--:(hmac-sha-1-12) {crypto-hmac-sha-1-12}?
      |     |     |  +--rw hmac-sha1-12?       empty
      |     |     +--:(aes-cmac-prf-128) {aes-cmac-prf-128}?
      |     |     |  +--rw aes-cmac-prf-128?   empty
      |     |     +--:(md5)
      |     |     |  +--rw md5?                empty
      |     |     +--:(sha-1)
      |     |     |  +--rw sha-1?              empty
      |     |     +--:(hmac-sha-1)
      |     |     |  +--rw hmac-sha-1?         empty
      |     |     +--:(hmac-sha-256)
      |     |     |  +--rw hmac-sha-256?       empty
      |     |     +--:(hmac-sha-384)
      |     |     |  +--rw hmac-sha-384?       empty
      |     |     +--:(hmac-sha-512)
      |     |        +--rw hmac-sha-512?       empty
      |     +--ro crypto-algorithm-state
      |        +--ro (algorithm)?
      |           +--:(hmac-sha-1-12) {crypto-hmac-sha-1-12}?
      |           |  +--ro hmac-sha1-12?       empty
      |           +--:(aes-cmac-prf-128) {aes-cmac-prf-128}?
      |           |  +--ro aes-cmac-prf-128?   empty
      |           +--:(md5)
```

```
      |            |  +--ro md5?                 empty
      |            +--:(sha-1)
      |            |  +--ro sha-1?               empty
      |            +--:(hmac-sha-1)
      |            |  +--ro hmac-sha-1?          empty
      |            +--:(hmac-sha-256)
      |            |  +--ro hmac-sha-256?        empty
      |            +--:(hmac-sha-384)
      |            |  +--ro hmac-sha-384?        empty
      |            +--:(hmac-sha-512)
      |               +--ro hmac-sha-512?        empty
      +--rw aes-key-wrap {aes-key-wrap}?
      |  +--rw enable?   boolean
      +--ro aes-key-wrap-state {aes-key-wrap}?
         +--ro enable?   boolean
```

4. Key Chain YANG Model

```
<CODE BEGINS> file "ietf-key-chain@2015-10-15.yang"
module ietf-key-chain {
  namespace "urn:ietf:params:xml:ns:yang:ietf-key-chain";
  // replace with IANA namespace when assigned
  prefix "key-chain";

  import ietf-yang-types {
    prefix "yang";
  }

  organization
    "IETF RTG (Routing) Working Group";
  contact
    "Acee Lindem - acee@cisco.com";

  description
    "This YANG module defines the generic configuration
     data for key-chain. It is intended that the module
     will be extended by vendors to define vendor-specific
     key-chain configuration parameters.

     Copyright (c) 2015 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 XXXX; see
        the RFC itself for full legal notices.";

     revision 2015-10-15 {
       description
         "Updated version, organization, and copyright.
          Added aes-cmac-prf-128 and aes-key-wrap features.";
       reference
         "RFC XXXX: A YANG Data Model for key-chain";
     }
     revision 2015-06-29 {
       description
         "Updated version. Added Operation State following
          draft-openconfig-netmod-opstate-00.";
       reference
         "RFC XXXX: A YANG Data Model for key-chain";
     }
     revision 2015-02-24 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for key-chain";
     }

     typedef key-chain-ref {
       type leafref {
         path "/key-chain:key-chains/key-chain:key-chain-list/"
            + "key-chain:name";
       }
       description
         "This type is used by data models that need to reference
          configured key-chains.";
     }

     /* feature list */
     feature hex-key-string {
       description
         "Support hexadecimal key string.";
     }

     feature accept-tolerance {
       description
         "To specify the tolerance or acceptance limit.";
     }

     feature independent-send-accept-lifetime {
       description
         "Support for independent send and accept key lifetimes.";
```

```
     }

     feature crypto-hmac-sha-1-12 {
       description
         "Support for TCP HMAC-SHA-1 12 byte digest hack.";
     }

     feature aes-cmac-prf-128 {
       description
         "Support for AES Cipher based Message Authentication Code
          Pseudo Random Function.";
     }

     feature aes-key-wrap {
       description
         "Support for Advanced Encryption Standard (AES) Key Wrap.";
     }

     /* groupings */
     grouping lifetime {
       description
         "Key lifetime specification.";
       choice lifetime {
         default always;
         description
           "Options for specifying key accept or send lifetimes";
         case always {
           leaf always {
             type empty;
               description
                 "Indicates key lifetime is always valid.";
           }
         }
         case start-end-time {
           leaf start-date-time {
             type yang:date-and-time;
             description "Start time.";
           }
           choice end-time {
             default infinite;
             description
               "End-time setting.";
             case infinite {
           leaf no-end-time {
                 type empty;
               description
                 "Indicates key lifetime end-time in infinite.";
               }
```

```
             }
             case duration {
               leaf duration {
                 type uint32 {
                   range "1..2147483646";
                 }
                 units seconds;
                 description "Key lifetime duration, in seconds";
               }
             }
             case end-date-time {
               leaf end-date-time {
                 type yang:date-and-time;
                 description "End time.";
               }
             }
           }
         }
       }
     }

    grouping crypto-algorithm-types {
        description "Cryptographic algorithm types.";
      choice algorithm {
        description
            "Options for cryptographic algorithm specification.";
        case hmac-sha-1-12 {
            if-feature crypto-hmac-sha-1-12;
            leaf hmac-sha1-12 {
                type empty;
                description "The HMAC-SHA1-12 algorithm.";
            }
        }
        case aes-cmac-prf-128 {
            if-feature aes-cmac-prf-128;
            leaf aes-cmac-prf-128 {
                type empty;
                description "The AES-CMAC-PRF-128 algorithm - required
                             by RFC 5926 for TCP-AO key derivation
                             functions.";
            }
        }
        case md5 {
            leaf md5 {
                type empty;
                description "The MD5 algorithm.";
            }
        }
```

```
        case sha-1 {
            leaf sha-1 {
                type empty;
                description "The SHA-1 algorithm.";
            }
        }
        case hmac-sha-1 {
            leaf hmac-sha-1 {
                type empty;
                description "HMAC-SHA-1 authentication algorithm.";
            }
        }
        case hmac-sha-256 {
            leaf hmac-sha-256 {
                type empty;
                description "HMAC-SHA-256 authentication algorithm.";
            }
        }
        case hmac-sha-384 {
            leaf hmac-sha-384 {
                type empty;
                description "HMAC-SHA-384 authentication algorithm.";
            }
        }
        case hmac-sha-512 {
            leaf hmac-sha-512 {
                type empty;
                description "HMAC-SHA-512 authentication algorithm.";
            }
        }
      }
    }

     grouping key-chain {
       description
         "key-chain specification grouping.";
       leaf name {
         type string;
         description "Name of the key-chain.";
       }

       leaf name-state {
         type string;
         config false;
         description "Configured name of the key-chain.";
       }

       container accept-tolerance {
```

```
          if-feature accept-tolerance;
          description
            "Tolerance for key lifetime acceptance (seconds).";
          leaf duration {
            type uint32;
            units seconds;
            default "0";
            description
              "Tolerance range, in seconds.";
          }
        }

        container accept-tolerance-state {
          config false;
          description
            "Configured tolerance for key lifetime
             acceptance (seconds).";
          leaf duration {
            type uint32;
            description
              "Configured tolerance range, in seconds.";
          }
        }

        list key-chain-entry {
          key "key-id";
          description "One key.";
          leaf key-id {
            type uint64;
            description "Key ID.";
          }
          leaf key-id-state {
            type uint64;
            config false;
            description "Configured Key ID.";
          }
          container key-string {
            description "The key string.";
            choice key-string-style {
               description
                 "Key string styles";
               case keystring {
                 leaf keystring {
                   type string;
                   description "Key string in ASCII format.";
                 }
               }
               case hexadecimal {
```

```
                if-feature hex-key-string;
                leaf hexadecimal-string {
                  type yang:hex-string;
                  description
                    "Key in hexadecimal string format.";
                }
              }
            }
         }
         container lifetime {
           description "Specify a key's lifetime.";
           choice lifetime {
             description
               "Options for specification of send and accept
                lifetimes.";
             case send-and-accept-lifetime {
               description
                 "Send and accept key have the same lifetime.";
               container send-accept-lifetime {
                 uses lifetime;
                 description
                   "Single lifetime specification for both send and
                    accept lifetimes.";
               }
             }
             case independent-send-accept-lifetime {
               if-feature independent-send-accept-lifetime;
               description
                 "Independent send and accept key lifetimes.";
               container send-lifetime {
                 uses lifetime;
                 description
                   "Separate lifetime specification for send
                    lifetime.";
               }
               container accept-lifetime {
                 uses lifetime;
                 description
                   "Separate lifetime specification for accept
                    lifetime.";
               }
             }
           }
         }
         container lifetime-state {
           config false;
           description "Configured key's lifetime.";
           container send-lifetime {
```

```
            uses lifetime;
            description
              "Configured send-lifetime.";
          }
          leaf send-valid {
            type boolean;
            description
              "Status of send-lifetime.";
          }
          container accept-lifetime {
            uses lifetime;
            description
              "Configured accept-lifetime.";
          }
          leaf accept-valid {
            type boolean;
            description
              "Status of accept-lifetime.";
          }
        }
        container crypto-algorithm {
          uses crypto-algorithm-types;
          description "Cryptographic algorithm associated with key.";
        }
        container crypto-algorithm-state {
          config false;
          uses crypto-algorithm-types;
          description "Configured cryptographic algorithm.";
        }
      }
    }

    container key-chains {
      list key-chain-list {
        key "name";
        description
          "List of key-chains.";
        uses key-chain;
      }
      container aes-key-wrap {
        if-feature aes-key-wrap;
        leaf enable {
          type boolean;
          default false;
          description
            "Enable AES Key Wrap encryption.";
        }
        description
```

```
          "AES Key Wrap password encryption.";
      }
      container aes-key-wrap-state {
        if-feature aes-key-wrap;
        config false;
        leaf enable {
          type boolean;
          description "AES Key Wrap state.";
        }
        description "Status of AES Key Wrap.";
      }
      description "All configured key-chains for the device.";
    }
  }
  <CODE ENDS>
```

5. Relationship to other Work

6. Security Considerations

This document enables the automated distribution of industry standard key chains using the NETCONF [NETCONF] protocol. As such, the security considerations for the NETCONF protocol are applicable. Given that the key chains themselves are sensitive data, it is RECOMMENDED that the NETCONF communication channel be encrypted. One way to do accomplish this would be to invoke and run NETCONF over SSH as described in [NETCONF-SSH].

When configured, the key-strings can be encrypted using the AES Key Wrap algorithm [AES-KEY-WRAP]. The AES key-encryption key (KEK) is not included in the YANG model and must be set or derived independent of key-chain configuration.

The key strings are not included in the operational state. This is a practice carried over from SNMP MIB modules and is an area for further discussion.

7. IANA Considerations

This document registers a URI in the IETF XML registry [XML-REGISTRY]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-key-chain

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [YANG].

name: ietf-acl namespace: urn:ietf:params:xml:ns:yang:ietf-key-chain prefix: ietf-key-chain reference: RFC XXXX

8. References

8.1. Normative References

[NETCONF] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[NETCONF-SSH] Wasserman, M., "Using NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC-KEYWORDS] Bradner, S., "Key words for use in RFC’s to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[XML-REGISTRY] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[YANG] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

8.2. Informative References

[AES-KEY-WRAP] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, August 2009.

[CRYPTO-KEYTABLE] Housley, R., Polk, T., Hartman, S., and D. Zhang, "Table of Cryptographic Keys", RFC 7210, April 2014.

[IAB-REPORT] Andersson, L., Davies, E., and L. Zhang, "Report from the IAB workshop on Unwanted Traffic March 9-10, 2006", RFC 4948, August 2007.

[NTP-PROTO]
Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[OSPFV3-AUTH]
Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, March 2014.

[TCP-AO-ALGORITHMS]
Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", draft-chen-rtg-key-table-yang-00.txt (work in progress), June 2010.

[YANG-CRYPTO-KEYTABLE]
Chen, I., "YANG Data Model for RFC 7210 Key Table", draft-chen-rtg-key-table-yang-00.txt (work in progress), March 2015.

Appendix A. Acknowledgments

The RFC text was produced using Marshall Rose’s xml2rfc tool.

Thanks to Brian Weis for fruitful discussions on security requirements.

Authors’ Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Yingzhen Qu
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA

Email: yiqu@cisco.com

Derek Yeung
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA

Email: myeung@cisco.com

Ing-Wher Chen
Ericsson

Email: ing-wher.chen@ericsson.com

Jeffrey Zhang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: zzhang@juniper.net

Yi Yang
Cisco Systems
7025 Kit Creek Road
Research Triangle Park, NC 27709
USA

Email: yiya@cisco.com

Internet Draft I. Chen
<draft-chen-rtgwg-key-table-yang-02.txt> Ericsson
Intended Status: Standards Track
Expires in 6 months November 2, 2015

YANG Data Model for RFC 7210 Key Table
<draft-chen-rtgwg-key-table-yang-02.txt>

Status of this Memo

Copyright Notice

Abstract

RFC 7210 defines a key table that consists of cryptographic keys that stores information for many different types of routing protocols to ensure message security. This document defines a YANG data model that represents the key table defined in RFC 7210, with the information necessary for YANG and NETCONF to provide routing protocol authentication.

Table of Contents

1. Introduction

[RFC7210] defines a standards track key table that is used to store information for routing protocol authentication, including key values, cryptographic algorithms, timing attributes, and their relationships to allow different routing protocols to perform key selection, authentication, and smooth key rollover that ultimate provides routing protocol message security. This document defines a YANG [RFC6020] data model that corresponds to [RFC7210] and enables the use of NETCONF [RFC6241] and YANG to manage routing protocol authentication data.

An earlier version of the key table YANG model [I-D.chen-rtg-key-table-yang] augments from the key-chain YANG model [I-D.acee-rtg-yang-key-chain]. However, because the key-chain YANG model organizes keys in groups, which is different from the key definitions in [RFC7210], this document proposes a new key-table YANG model that is structurally more consistent with the key database defined in [RFC7210].

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.2. Tree Diagram

A simplified graphical representation of the data model is presented in Section 2.

The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.

- o Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.

- o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" state data (read-only).

- o Symbols after data node names: "?" means an optional node and "*" denotes a "list" or "leaf-list".

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Design of the Data Model

This data model is based on the key table defined in [RFC7210]. Because [RFC7210] defines a table that allows protocols to customize certain fields, the data model comes in two parts, the base model and the customizable part. The base model defines the attributes in [RFC7210] that are immutable across all protocols, such as admin-key-name and send-lifetime-start. The customizable part defines the attributes that are different across protocols, such as the local-key-name, peer-key-name, and peers field. These customizable attributes are defined as YANG grouping statements that can be used and incorporated into protocol-specific key table modules.

2.1. Base Model

[RFC7210] defines a single table in which the rows represent the individual key entries of the key table. As such, the base model consists of one single top-level container named "key-table", in which a YANG list named "security-association-entry" is defined. The entries in the "security-association-entry" YANG list represents the rows that correspond to the cryptographic keys in [RFC7210] key table.

The base model further defines each "security-association-entry" to consist of attributes that are common across all protocols:

o admin-key-name

o key

o interfaces

o send-lifetime-start

o send-lifetime-end

o accept-lifetime-start

o accept-lifetime-end

Additionally, the base model also defines two attributes, "peers" and "protocol-specific-info", as placeholders left for each protocol to define, as prescribed by [RFC7210].

```
module: ietf-key-table
   +--rw key-table
      +--rw security-association-entry* [admin-key-name]
         +--rw admin-key-name            string
         +--rw peers
         +--rw interfaces
         |  +--rw (interface-options)
         |     +--:(all-interfaces)
         |     |  +--rw all?         empty
         |     +--:(interface-list)
         |        +--rw interface*   if:interface-ref
         +--rw protocol                  identityref
         +--rw protocol-specific-info
         +--rw key                       yang:hex-string
         +--rw send-lifetime-start       lifetime-type
         +--rw send-lifetime-end         lifetime-type
         +--rw accept-lifetime-start     lifetime-type
         +--rw accept-lifetime-end       lifetime-type
```

2.2. Protocol Customization

Besides the attributes defined in the base model, for each row in the key table, [RFC7210] also defines the following attributes for which the format and range of valid values are protocol-specific.

o local key name

o peer key name

o key derivation function

o cryptographic algorithm

o direction

After a routing protocol augments the base model to incorporate the attributes above, the result is a key table with all the necessary attributes for routing protocol authentication, as shown in the key table below with RSVP customization.

```
module: ietf-key-table
   +--rw key-table
      +--rw security-association-entry* [admin-key-name]
         +--rw admin-key-name            string
         +--rw interfaces
         |  +--rw (interface-options)
         |     +--:(all-interfaces)
         |     |  +--rw all?         empty
         |     +--:(interface-list)
         |        +--rw interface*   if:interface-ref
         +--rw protocol                  identityref
         +--rw protocol-specific-info
         +--rw key                       yang:hex-string
         +--rw send-lifetime-start       lifetime-type
         +--rw send-lifetime-end         lifetime-type
         +--rw accept-lifetime-start     lifetime-type
         +--rw accept-lifetime-end       lifetime-type
         +--rw peers
augment
/keytable:key-table/keytable:security-association-entry +
/keytable:peers:
         |  +--rw rsvp-security-association?   leafref
augment
/keytable:key-table/keytable:security-association-entry:
         +--rw kdf?         key-derivation-function-type
         +--rw alg-id?      cryptographic-algorithm-type
         +--rw direction?   enumeration

module: example-rsvp-key-table
   +--rw example-rsvp-key-table
      +--rw rsvp-security-association-entry* [name]
         +--rw name                   string
         +--rw rsvp-local-key-name?   uint64
         +--rw rsvp-peer-key-name?    uint64
```

3. Key-chain vs. Key-table Comparison

The key-chain YANG model also proposes a YANG model for key management. This section compares the key-chain model and the key-table model proposed in this document.

3.1. Organization

The key-chain YANG model groups several keys into a single key chain. It is the key chain that is referenced and applied by routing protocols. Consequently, the key-chain YANG model defines a hierarchical database, which consists of a top-level key-chain database, and each key-chain database consists of the actual keys that are used by a protocol. A routing protocol that requires encryption or authentication must reference a key-chain instead of the individual keys.

In contrast, [RFC7210] defines a single flat database of keys and their attributes. [RFC7210] does not require that an implementation of key management explicitly group a set of keys into a separate entity that routing protocols reference and use. Consequently, the key-table base model defined in this document presents a flat view of the key database.

For routing protocol customization, the key-table base model can also be adapted to hierarchical key management as described in the key-chain YANG model. Section 5 provides an example of adapting a hierarchical key management model into the key-table model.

3.2. Missing Attributes

The key-chain YANG model defines only a subset of key attributes defined in [RFC7210]. Consequently, the key-chain YANG model can only support specific types of deployments requiring authentication. A list of key attributes defined in [RFC7210] and in this document, but not defined in [key-chain], are as follows:

- Peer key name
- Interfaces
- Protocol
- Key derivation function
- Direction

In addition to the list above, [key-chain] also does not define attributes that [RFC7210] defined but left the detailed definitions to individual routing protocols. Similar to [RFC7210], this document defines YANG constructs for these attributes and intends for routing protocols to provided the details. The attributes in question are as follows:

o Peers

o ProtocolSpecificInfo

4. YANG Module

```
<CODE BEGINS> file "ietf-key-table@2015-08-28.yang"

module ietf-key-table {
  namespace "http://www.example.com/ietf-key-table";
  prefix "keytable";

  import ietf-yang-types {
    prefix "yang";
  }

  import ietf-routing {
    prefix "rt";
  }

  import ietf-interfaces {
    prefix "if";
  }

  organization
    "Ericsson";

  contact
    "I. Chen - ing-wher.chen@ericsson.com";

  description
    "A key table YANG data model based on RFC 7210";

  revision 2015-08-28 {
    description
      "Revision 3.  " +
      "Making RFC 7210 a global generic table.";
      reference "RFC 7210";
  }
```

```
revision 2015-06-29 {
  description
    "Revision 2.";
  reference "RFC 7210";
}

/* Identities */

identity key-derivation-function {
  description
    "Base identity from which key derivation function " +
    "identities are derived";
}

identity kdf-none {
  base key-derivation-function;
  description
    "This identity represents a cryptographic key that is" +
    "used directly, without a key derivation function.";
}

identity kdf-aes-128-cmac {
  base key-derivation-function;
  description
    "This identity represents the key derivation function that " +
    "uses AES-CMAC using 128-bit keys (RFC 4493).";
}

identity kdf-hmac-sha-1 {
  base key-derivation-function;
  description
    "This identity represents the key derivation function that " +
    "uses HMAC using the SHA-1 hash (RFC 2104).";
}

identity cryptographic-algorithm {
  description
    "Base identity from which cryptographic algorithms " +
    "are derived";
}

identity algid-aes-128-cmac {
  base cryptographic-algorithm;
  description
    "This identity represents the cryptographic algorithm " +
    "AES-CMAC using 128-bit keys (RFC 4493).";
}
```

```
     identity algid-aes-128-cmac-92 {
       base cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "AES-128-CMAC truncated to 96 bits (RFC 5926).";
     }

     identity algid-hmac-sha-1-96 {
       base cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "HMAC SHA-1 truncated to 96 bits (RFC 2104).";
     }

     identity all-routing-protocols {
       base rt:routing-protocol;
       description
         "All routing protocols";
     }

     /* Typedefs */
     typedef routing-protocol-type {
       type identityref {
         base rt:routing-protocol;
       }
       description
         "This type identifies the routing protocol";
     }

     typedef key-derivation-function-type {
       type identityref {
         base key-derivation-function;
       }
       description
         "This type identifies the key derivation function";
     }

     typedef cryptographic-algorithm-type {
       type identityref {
         base cryptographic-algorithm;
       }
       description
         "This type identifies the cryptographic algorithm";
     }

     typedef lifetime-type {
       type string {
         pattern '{4}{2}{2}{2}{2}Z';
```

```
       }
       description
         "This type identifies a time in the format YYYYMMDDHHSSZ, " +
         "where the first four digits specify the year, " +
         "the next two digits specify the month, " +
         "the next two digits specify the day, " +
         "the next two digits specify the hour, " +
         "the next two digits specify the second, " +
         "ending with the letter 'Z' as a clear indication " +
         "that the time is in Coordinated Universal Time (UTC).";
     }

     /* Groupings */

     grouping key-properties-grp {
       description
         "A grouping that to specify the properties of a key.  " +
         "This is defined as a grouping so that different " +
         "routing protocols can further refine the values of " +
         "each individual key properties.";
       leaf kdf {
         type key-derivation-function-type;
         description
           "Specify the key derivation function to be used " +
           "with this key.";
       }
       leaf alg-id {
         type cryptographic-algorithm-type;
         description
           "Specify the cryptographic algorithm to be used" +
           "with this key.";
       }
       leaf direction {
         type enumeration {
           enum "in" {
             description
               "The key is used for inbound traffic.";
           }
           enum "out" {
             description
               "The key is used for outbound traffic.";
           }
           enum "both" {
             description
               "The key is used for both inbound and outbound " +
               "traffic.";
           }
           enum "disabled" {
```

```
            description
              "The key is disabled and cannot be used for " +
              "either inbound or outbound traffic.";
          }
        }
        description
          "The value of the direction must be one of 'in', 'out', " +
          "'both', and 'disabled'.  The actual allowed value " +
          "is left for routing protocols to define.";
      }
    }

    /* The key-table model */

    container key-table {
      description
        "The key table of all managed cryptographic keys " +
        "of a device.";
      list security-association-entry {
        key "admin-key-name";
        description
          "A key table entry that specifies a key " +
          "and its attributes.";
        leaf admin-key-name {
          type string;
          description
            "A human-readable string that identifies the key.";
        }
        container peers {
          description
            "Specify the peer systems that also have this key " +
            "in their database.  The format of this field is " +
            "left to protocols to define.";
        }
        container interfaces  {
          description
            "Specify the interfaces to which they key may be applied.";
          choice interface-options {
            mandatory true;
            description
              "The option to apply this key to all interfaces or " +
              "to a pre-defined list of interfaces.";
            case all-interfaces {
              leaf all {
                type empty;
                description
                  "This key applies to all interfaces.";
              }
```

```
            }
            case interface-list {
              leaf-list interface {
                type if:interface-ref;
                description
                  "This key applies to the identified interfaces.";
              }
            }
          }
        }
        leaf protocol {
          type identityref {
            base rt:routing-protocol;
          }
          mandatory true;
          description
            "Specify a single routing protocol where this key " +
            "may be used to provide cryptographic protection.";
        }
        container protocol-specific-info {
          description
            "This field contains protocol-specified information " +
            "that maybe useful for a protocol to apply the key " +
            "correctly.  This field is left for each protocol " +
            "to define.";
        }
        leaf key {
          type yang:hex-string;
          mandatory true;
          description
            "The key";
        }
        leaf send-lifetime-start {
          type lifetime-type;
          mandatory true;
          description
            "Specify the earliest date and time at which this key " +
            "should be considered for use when sending traffic.";
        }
        leaf send-lifetime-end {
          type lifetime-type;
          mandatory true;
          description
            "Specify the latest date and time at which this key " +
            "should be considered for use when sending traffic.";
        }
        leaf accept-lifetime-start {
          type lifetime-type;
```

```
          mandatory true;
          description
            "Specify the earliest date and time at which this key " +
            "should be considered for processing received traffic.";
        }
        leaf accept-lifetime-end {
          type lifetime-type;
          mandatory true;
          description
            "Specify the latest date and time at which this key " +
            "should be considered for processing received traffic.";
        }
      }
    }
  }

  <CODE ENDS>
```

5. Usage

A routing protocol that requires cryptographic key authentication should customize the key table base model such that the resulting YANG modules describe all the necessary attributes and also the valid values of these attributes.

An an example, the customized example-ospf-key-table YANG module below augments the key table base model to include "key-derivation-function", "cryptofgraphic-algorithm", and "direction" attributes, as well as specifying the valid valuesfor those attributes. The module below further uses the "peers" field in the base key table to reference keys that are organized into key chains as expected by the existing OSPF YANG module [I-D.ospf-yang].

```
module example-ospf-key-table {
  namespace "http://www.example.com/example-ospf-key-table";
  prefix "ospf-keytable";

  import ietf-key-table {
    prefix "keytable";
  }

  import ietf-routing {
    prefix "rt";
  }

  organization
    "Ericsson";

  contact
```

```
       "I. Chen - ing-wher.chen@ericsson.com";

     description
       "OSPF’s customized key table";

     revision 2015-08-28 {
       description "Initial revision";
       reference "";
     }

     /* Identities */

     identity ospf-cryptographic-algorithm {
       base keytable:cryptographic-algorithm;
       description
         "Base identity from which OSPF cryptographic algorithm " +
         "identities are derived";
     }

     identity ospf-algid-md5 {
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "MD5";
     }

     identity ospf-algid-hmac-md5 {
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm "  +
         "HMAC-MD5";
     }

     identity ospf-algid-sha-1 {
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "SHA-1";
     }

     identity ospf-algid-hmac-sha-1 {
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "HMAC-SHA-1";
     }

     identity ospf-algid-hmac-sha-1-12 {
```

```
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "HMAC-SHA-1-12";
     }

     identity ospf-algid-hmac-sha-256 {
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "HMAC-SHA-256";
     }

     identity ospf-algid-hmac-sha-384 {
       base ospf-cryptographic-algorithm;
       description
         "This identity represents the cryptographic algorithm " +
         "HMAC-SHA-384";
     }

     identity ospf-algid-hmac-sha-512 {
       base ospf-cryptographic-algorithm;
       description
         "This identity repesents the cryptographic algorithm " +
         "HMAC-SHA-512";
     }

     /* Typedef */

     typedef ospf-cryptographic-algorithm-type {
       type identityref {
         base ospf-cryptographic-algorithm;
       }
       description
         "This type identifies the cryptographic algorithm";
     }

     augment "/keytable:key-table/keytable:security-association-entry" {
       when "keytable:protocol == 'rt:ospfv2' or " +
            "keytable:protocol == 'rt:ospfv3'" {
         description
           "Applies only to OSPFv2 and OSPFv3.";
       }
       uses keytable:key-properties-grp {
         refine "kdf" {
           must ". == 'keytable:kdf-none'" {
             description
               "KDF is not used.";
```

```
          }
        }
        refine "alg-id" {
          must ". == 'ospf-algid-md5' or " +
               ". == 'ospf-algid-hmac-md5' or " +
               ". == 'ospf-algid-sha-1' or " +
               ". == 'ospf-algid-hmac-sha-1' or " +
               ". == 'ospf-algid-hmac-sha-1-12' or " +
               ". == 'ospf-algid-hmac-sha-256' or " +
               ". == 'ospf-algid-hmac-sha-384' or " +
               ". == 'ospf-algid-hmac-sha-512' or " {
            description
              "Only ospf-cryptographic-algorithms are valid.";
          }
        }
        refine "direction" {
          must ". == 'keytable:both'" {
            description
              "Key applies to both directions.";
          }
        }
      }
      description
        "Customize OSPF protocol specific attributes";
    }

    augment "/keytable:key-table" +
            "/keytable:security-association-entry" +
            "/keytable:peers" {
      when "../keytable:protocol == 'rt:ospfv2' or " +
           "../keytable:protocol == 'rt:ospfv3'" {
        description
          "Applies only to OSPFv2 and OSPFv3.";
      }
      description
        "Reference to the apropriate key chain";
      leaf key-chain {
        type leafref {
          path "/example-ospf-key-chains/ospf-key-chain/name";
        }
        description
          "The name of the key chain";
      }
      leaf security-association {
        type leafref {
          path "/example-ospf-key-chains" +
               "/ospf-key-chain[name = current()/../key-chain]" +
               "/ospf-security-association-entry/name";
```

```
        }
        description
          "The security association within the key chain";
      }
    }

    container example-ospf-key-chains {
      description
        "A container of OSPF key chains modeled after " +
        "ietf-key-chains";
      list ospf-key-chain {
        key "name";
        leaf name {
          type string;
          description
            "Name of the key chain";
        }
        list ospf-security-association-entry {
          key "name";
          leaf name {
            type string;
            description
              "The name of the security assocation";
          }
          leaf ospf-local-key-name {
            type uint8;
            mandatory true;
            description
              "The 8-bit key ID for sending a message, " +
              "as defined in RFC 2328 Appendix D.3";
          }
          leaf ospf-peer-key-name {
            type leafref {
              path "../ospf-local-key-name";
            }
            description
              "The 8-bit key ID when receiving a message, " +
              "as defined in RFC 2328 Appendix D.3. " +
              "Because OSPF uses the same key for sending " +
              "and receiving, the value of this leaf " +
              "should be identical to the value of " +
              "ospf-local-key-name.";
          }

          description
            "An OSPF security association, i.e. key";
        }
        description
```

```
           "An OSPF key chain";
       }
     }
   }
```

6. Security Consideration.

TBD.

7. IANA Considerations

TBD.

8. References

8.1. Normative References

[RFC7210] Housley, R., Polk, T., Hartman, S., and D. Zhang, "Database of Long-Lived Symmetric Cryptographic Keys", RFC 7210, April 2014, <http://www.rfc-editor.org/info/rfc7210>.

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

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2. Informative References

[I-D.acee-rtg-yang-key-chain] Lindem, A., Qu, Y., Yeung, D., Chen, I., Zhang, J., and Y. Yang, "Key Chain YANG Data Model", draft-acee-rtg-yang-key-chain-09 (work in progress), October 2015.

[I-D.ospf-yang] Yeung, D., Qu, Y., Zhang, J., Bogdanovic, D., and K. Sreenivasa, "draft-ietf-ospf-yang-02 (work in progress)", September 2015.

Author's Address

I. Chen
Ericsson
Email: ing-wher.chen@ericsson.com

NETCONF Working Group K. Watsen
Internet-Draft Juniper Networks
Intended status: Standards Track J. Schoenwaelder
Expires: April 11, 2016 Jacobs University Bremen
October 9, 2015

NETCONF Server and RESTCONF Server Configuration Models
draft-ietf-netconf-server-model-08

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:

o draft-ietf-netconf-restconf

o draft-ietf-netconf-call-home

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

o "VVVV" --> the assigned RFC value for this draft

o "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf

o "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:

- "2015-10-09" --> the publication date of this draft

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

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

Status of This Memo

Copyright Notice

Table of Contents

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.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o 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-keychain|
                     +-------------+
                         ^    ^
                         |    |
              <leafref>  |    |  <leafref>
            +------------+    +------------+
            |                              |
     +---------------+            +------------------+
     |ietf-ssh-server|            | ietf-tls-server  |
     +---------------+            +------------------+
       ^                               ^       ^
       | <uses>                        |       |
       |            <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 Keychain Model

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

- o A semi-configurable list of private keys, each with one or more associated certificates. Though private keys can only be created via an RPC (see bullet #3 below), the entries of the list may be renamed and have certificates associated with them after creation.

- o 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).

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

o 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 set using a standard configuration change request (e.g., <edit-config>).

4.1.1. Tree Diagram

```
module: ietf-keychain
   +--rw keychain
      +--rw private-keys
      |  +--rw private-key* [name]
      |  |  +--rw name                                     string
      |  |  +--ro algorithm?                               enumeration
      |  |  +--ro key-length?                              uint32
      |  |  +--ro public-key?                              string
      |  |  +--rw certificates
      |  |  |  +--rw certificate* [name]
      |  |  |     +--rw name     string
      |  |  |     +--rw chain?   binary
      |  |  +---x generate-certificate-signing-request
      |  |     +---w input
      |  |     |  +---w subject       binary
      |  |     |  +---w attributes?   binary
      |  |     +--ro output
      |  |        +--ro certificate-signing-request    binary
      |  +---x generate-private-key
      |     +---w input
      |        +---w name          string
      |        +---w algorithm     enumeration
      |        +---w key-length?   uint32
      +--rw trusted-certificates* [name]
         +--rw name                    string
         +--rw description?            string
         +--rw trusted-certificate* [name]
            +--rw name           string
            +--rw certificate?   binary
```

4.1.2. Example Usage

The following example illustrates the "generate-private-key" RPC in use with the RESTCONF protocol and JSON encoding.

REQUEST

```
[’\’ line wrapping added for formatting only]

POST https://example.com/restconf/data/ietf-keychain:keychain/\
private-keys/generate-private-key HTTP/1.1
HOST: example.com
Content-Type: application/yang.operation+json
```



```
{
  "ietf-keychain:input" : {
    "name" : "ex-key-sect571r1",
    "algorithm" : "sect571r1"
  }
}
```


RESPONSE

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

The following example illustrates the action statement "generate-certificate-signing-request" action in use with the NETCONF protocol.

REQUEST

```
<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:yang:ietf-keychain">
      <private-keys>
        <private-key>
          <name>ex-key-sect571r1</name>
          <generate-certificate-signing-request>
            <subject>
             cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2R
             manZvO3NkZmJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNlmO
             Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmR6Zgo=
            </subject>
            <attributes>
             bwtakWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvut4
             arnZvO3NkZmJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYm
```

```
                Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmC6Rhp=
              </attributes>
            </generate-certificate-signing-request>
          </private-key>
        </private-keys>
      </keychain>
    </action>
  </rpc>
```


RESPONSE

```
  <rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <certificate-signing-request
       xmlns="urn:ietf:params:xml:ns:yang:ietf-keychain">
       LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
       0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
       FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
       GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
       diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
       KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
       El1QWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJTdGZQY3N0Zk1KT1
       FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAvU25FcFE0TnV
       bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
       URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
       ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
       mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
       RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
       rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
       TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
       c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
       SWHgzZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
     </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 configured back onto the server. Lastly, this example shows that three lists of trusted certificates having been configured.

```
<keychain xmlns="urn:ietf:params:xml:ns:yang:ietf-keychain">
```

```
<!-- private keys and associated certificates -->
<private-keys>
  <private-key>
    <name>ex-key-sect571r1</name>
    <algorithm>sect571r1</algorithm>
    <public-key>
      cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
      mJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYmZ2aXNiZGZpYmhzZG87Zm
      JvO3NkZ25iO29pLmR6Zgo=
    </public-key>
    <certificates>
      <certificate>
        <name>ex-key-sect571r1-cert</name>
        <data>
          LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
          0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
          FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
          GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
          diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
          KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
          El1QWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJTdGZQY3N0Zk1KT1
          FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAvU25FcFE0TnV
          bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
          URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
          ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
          mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
          RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
          rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
          TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
          c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
          SWHgzZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
        </data>
      </certificate>
    </certificates>
  </private-key>
</private-keys>

<!-- trusted netconf/restconf client certificates -->
<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-certificate>
    <name>George Jetson</name>
    <certificate>
```

```
          QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ
          MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
          25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2
          RV0JCU2t2MXI2SFNHeUFUVkpwSmYyOWtXbUU0NEo5akJrQmdOVkhTTUVY
          VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
          UxNQWtHQTFVRUJoTUNWVk14RURBT0JnTlZCQW9UQjJWNApZVzF3YkdVeE
          V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOOMVpYS0NDUUNVRHBNSll6UG8zREF
          NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
          Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGGKYlhCc1pTN
          WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
          xWVE1SQXdEZ1lEVlFRSwpFd2RsZUdGdGNHeGxNUk13RVFZRFZRUURFd3B
          EVWt3Z1NYTnpkV1Z5TUEwR0NTcUdTSWIzRFFFQkJRVUFBNEdCCkFFc3BK
          WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
          TQzcjFZSjk0M1FQLzV5eGUKN2QxMkxCV0dxUjUrbEl5N01YL21ka2M4al
          zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
          LS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
        </certificate>
      </trusted-certificate>
      <trusted-certificate>
        <name>Fred Flinstone</name>
        <certificate>
          VlEVlFRREV3Vm9ZWEJ3ZVRDQm56QU5CZ2txaGtpRzl3MEJBUUVGQUFPQm
          pRQXdnWWtDCmdZRUE1RzRFSWZsS1p2bDlXTW44eUhyM2hObUFRaUhVUzV
          rRUpPQy9hSFA3eGJXQW1ra054ZStUa2hrZnBsL3UKbVhsTjhSZUd1ODhG
          NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPPCk1CMEdBMVVkRGd
          VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
          V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOOMVpYS0NDUUNVRHBNSll6UG8zREF
          NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
          Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGGKYlhCc1pTN
          WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
          xWVE1SQXdEZ1lEVlFRSwpFd2RsZUdGdGNHeGxNUk13RVFZRFZRUURFd3B
          EVWt3Z1NYTnpkV1Z5TUEwR0NTcUdTSWIzRFFFQkJRVUFBNEdCCkFFc3BK
          WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
          lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk
          zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
          QWtUOCBDRVUUZJ0RUF==
        </certificate>
      </trusted-certificate>
    </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>
            WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
            lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk
            zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
            NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd
            VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
            V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF
            NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
            Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGGKYlhCc1pTN
            WpiMjB2WlhoaGJJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
            QmdOVkJBWVRBbFZUVVJBd0RnWURWUVFLRXdkbApsR0Z0Y0d4bE1RNHdEQ
            MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
            25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2
            RJSUJQFRStS0Cg==
          </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-certificate>
          <name>ex-certificate-authority</name>
          <certificate>
            NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd
            VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
            V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF
            Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGGKYlhCc1pTN
            QmdOVkJBWVRBbFZUVVJBd0RnWURWUVFLRXdkbApsR0Z0Y0d4bE1RNHdEQ
            MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
            NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
            WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
            lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk
            zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
            25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2
            WpiMjB2WlhoaGJJYQnNaUzVqY215aU9L=
          </certificate>
        </trusted-certificate>
      </trusted-certificates>
```

```
   </keychain>
```

4.1.3. YANG Model

```
   <CODE BEGINS> file "ietf-keychain@2015-10-09.yang"

   module ietf-keychain {
     yang-version 1.1;

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

     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 "2015-10-09" {
```

```
       description
        "Initial version";
       reference
        "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                   Models";
     }

     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 enumeration {
               enum rsa { description "TBD"; }
               enum dsa { description "TBD"; }
               enum secp192r1 { description "TBD"; }
               enum sect163k1 { description "TBD"; }
               enum sect163r2 { description "TBD"; }
               enum secp224r1 { description "TBD"; }
               enum sect233k1 { description "TBD"; }
               enum sect233r1 { description "TBD"; }
               enum secp256r1 { description "TBD"; }
               enum sect283k1 { description "TBD"; }
               enum sect283r1 { description "TBD"; }
               enum secp384r1 { description "TBD"; }
               enum sect409k1 { description "TBD"; }
               enum sect409r1 { description "TBD"; }
               enum secp521r1 { description "TBD"; }
               enum sect571k1 { description "TBD"; }
               enum sect571r1 { description "TBD"; }
             }
             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 string;
            config false;
            description
              "The public-key matching the private key.";
          }
          container certificates {
            list certificate {
              key name;
              description
                "A certificate for this public key.";
              leaf name {
                type string;
                description
                  "An arbitrary name for the certificate.";
              }
              leaf chain {
                type binary;
                description
                  "The certificate itself, as well as an ordered
                   sequence of intermediate certificates leading
                   to a trust anchor, as specified by RFC 5246,
                   Section 7.4.2.";
                reference
                  "RFC 5246: The Transport Layer Security (TLS)
                             Protocol Version 1.2";
              }
            }
            description
              "A list of certificates for this public key.";
          }
          action generate-certificate-signing-request {
            description
              "Generates a certificate signing request structure for
               the associated private key using the passed subject
               and attribute values.";
            input {
              leaf subject {
                type binary;
                mandatory true;
```

```
                 description
                   "The distinguished name of the certificate subject
                    (the entity whose public key is to be certified).
                    This field is encoded the same as the 'subject'
                    field in the CertificationRequestInfo type defined
                    in RFC 2986, Section 4.1.";
                 reference
                   "RFC 2986: PKCS #10: Certification Request Syntax
                              Specification Version 1.7";
               }
               leaf attributes {
                 type binary;
                 description
                   "A collection of attributes providing additional
                    information about the subject of the certificate.
                    This field is encoded the same as the 'attributes'
                    field  in the CertificationRequestInfo type defined
                    in RFC 2986, Section 4.1.";
                 reference
                   "RFC 2986: PKCS #10: Certification Request Syntax
                              Specification Version 1.7";
               }
             }
             output {
               leaf certificate-signing-request {
                 type binary;
                 mandatory true;
                 description
                   "The certificate signing request to be signed by
                    a certificate authority.  This field is encoded
                    as the CertificationRequest type defined in
                    RFC 2986, Section 4.2.";
                 reference
                   "RFC 2986: PKCS #10: Certification Request Syntax
                              Specification Version 1.7";
               }
             }
           }
         }
         action generate-private-key {
           description
             "Generates 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 algorithm {
              type enumeration {
                enum rsa { description "TBD"; }
                enum dsa { description "TBD"; }
                enum secp192r1 { description "TBD"; }
                enum sect163k1 { description "TBD"; }
                enum sect163r2 { description "TBD"; }
                enum secp224r1 { description "TBD"; }
                enum sect233k1 { description "TBD"; }
                enum sect233r1 { description "TBD"; }
                enum secp256r1 { description "TBD"; }
                enum sect283k1 { description "TBD"; }
                enum sect283r1 { description "TBD"; }
                enum secp384r1 { description "TBD"; }
                enum sect409k1 { description "TBD"; }
                enum sect409r1 { description "TBD"; }
                enum secp521r1 { description "TBD"; }
                enum sect571k1 { description "TBD"; }
                enum sect571r1 { description "TBD"; }
              }
              mandatory true;
              description
                "The algorithm to be used.";
            }
            leaf key-length {
              type uint32;
              description
                "For algorithms that need a key length specified
                 when generating the key.";
            }
          }
        }
      }

      list trusted-certificates {
        key name;
        description
          "A list of lists of trusted certificates.";
        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 list of trusted certificates for a specific use.";
           leaf name {
             type string;
             description
               "An arbitrary name for this trusted certificate.";
           }
           leaf certificate {
             type binary;
             description
               "The binary certificate structure as specified by RFC
                5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
               ";
             reference
               "RFC 5246: The Transport Layer Security (TLS)
                          Protocol Version 1.2";
           }
         }
       }
     }
   }


   <CODE ENDS>
```

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/pri
vate-key/name
      |        +--:(certificate)
      |           +--rw certificate?   -> /kc:keychain/private-keys/pri
vate-key/certificates/certificate/name {ssh-x509-certs}?
      +--rw client-cert-auth {ssh-x509-certs}?
         +--rw trusted-ca-certs?       -> /kc:keychain/trusted-certific
ates/name
         +--rw trusted-client-certs?   -> /kc:keychain/trusted-certific
ates/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>
  </certificates>
  <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

```
<CODE BEGINS> file "ietf-ssh-server@2015-10-09.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-keychain {
    prefix kc;                       // RFC VVVV
    revision-date 2015-10-09;
  }

  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 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 "2015-10-09" {
       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 advertises
              when sending its ??? message.";
           reference
             "RFC ????: ...";
           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 name of a private-key in the keychain.";
             }
             leaf certificate {
               if-feature ssh-x509-certs;
               type leafref {
                 path "/kc:keychain/kc:private-keys/kc:private-key/"
                      + "kc:certificates/kc:certificate/kc:name";
               }
               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
           listens on.";
       }
       uses non-listening-ssh-server-grouping;
     }

     // RFC Editor: please remove the following container block
     //             when publishing this document as an RFC.
```

```
     container listening-ssh-server {
       description
         "This container is only present to enable ‘pyang‘
          tree diagram output, as a grouping by itself has
          no protocol accessible nodes to output.";

          uses listening-ssh-server-grouping;
     }


   }



   <CODE ENDS>
```

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/cert
ificates/certificate/name
      +--rw client-auth
         +--rw trusted-ca-certs?       -> /kc:keychain/trusted-certific
ates/name
         +--rw trusted-client-certs?   -> /kc:keychain/trusted-certific
ates/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@2015-10-09.yang"

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

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

  import ietf-inet-types {           // RFC 6991
    prefix inet;
  }
```

```
     import ietf-keychain {
       prefix kc;                       // RFC VVVV
       revision-date 2015-10-09;
     }

     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 "2015-10-09" {
       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.";
         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:certificates/kc:certificate/kc:name";
             }
             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 clients 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 TLTLS server
           listens on.";
       }
       uses non-listening-tls-server-grouping;
     }



     // RFC Editor: please remove the following container block
     //             when publishing this document as an RFC.
     container listening-tls-server {
       description
         "This container is only present to enable ‘pyang‘
          tree diagram output, as a grouping by itself has
          no protocol accessible nodes to output.";

          uses listening-tls-server-grouping;
     }


   }
```

<CODE ENDS>

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 ssh
       |        |     +--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
 rivate-keys/private-key/certificates/certificate/name {ssh-x509-certs}?
       |        |     +--rw client-cert-auth {ssh-x509-certs}?
       |        |        +--rw trusted-ca-certs?       -> /kc:keychain/t
```

```
rusted-certificates/name
      |        |        +--rw trusted-client-certs?   -> /kc:keychain/t
rusted-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
rivate-key/certificates/certificate/name
      |              +--rw client-auth
      |                 +--rw trusted-ca-certs?       -> /kc:keychain/t
rusted-certificates/name
      |                 +--rw trusted-client-certs?   -> /kc:keychain/t
rusted-certificates/name
      |                 +--rw cert-maps
      |                    +--rw cert-to-name* [id]
      |                       +--rw id             uint32
      |                       +--rw fingerprint    x509c2n:tls-fingerpr
int
      |                       +--rw map-type       identityref
      |                       +--rw name           string
      +--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
rivate-keys/private-key/name
            |  |     |        +--:(certificate)
            |  |     |           +--rw certificate?   -> /kc:keychain/p
rivate-keys/private-key/certificates/certificate/name {ssh-x509-certs}?
            |  |     +--rw client-cert-auth {ssh-x509-certs}?
            |  |        +--rw trusted-ca-certs?       -> /kc:keychain/t
rusted-certificates/name
            |  |        +--rw trusted-client-certs?   -> /kc:keychain/t
```

```
 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/p
 rivate-key/certificates/certificate/name
             |        +--rw client-auth
             |           +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw cert-maps
             |              +--rw cert-to-name* [id]
             |                 +--rw id             uint32
             |                 +--rw fingerprint    x509c2n:tls-fingerpr
 int
             |                 +--rw map-type       identityref
             |                 +--rw name           string
             +--rw connection-type
             |  +--rw (connection-type)?
             |     +--:(persistent-connection)
             |     |  +--rw persistent!
             |     |     +--rw idle-timeout?   uint32
             |     |     +--rw keep-alives
             |     |        +--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.

```
<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>
            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>
              <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-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>
          <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@2015-10-09.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 2015-10-09;
    }
    import ietf-tls-server {           // RFC VVVV
      prefix ts;
      revision-date 2015-10-09;
    }
```

```
   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 "2015-10-09" {
     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 5539: 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 {  // SHOULD WE REMOVE THIS ALTOGETHER?
        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
                       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";
                      }
                      units seconds;
                      default 30;
```

```
                      description
                       "Sets the amount of time in seconds after which
                if no data has been received from the SSH/TLS
                client, a SSH/TLS-level message will be sent
                        to test the aliveness of the SSH/TLS client.";
                    }
                    leaf max-attempts {
                      type uint8;
                      default 3;
                      description
                       "Sets the number of maximum number of sequential
                        keep-alive messages that can fail to obtain a
                        response from the SSH/TLS client before assuming
                        the SSH/TLS client is no longer alive.";
                    }
                  }
                }
              }
              case periodic-connection {
                container periodic {
                  presence true;
                  description
                   "Periodically connect to the NETCONF client, so that
                    the NETCONF client may deliver messages pending for
                    the NETCONF server.  The NETCONF client is expected
                    to close the connection when it is ready to release
                    it, thus starting the NETCONF server's timer until
                    next connection.";
                  leaf idle-timeout {
                    type uint16;
                    units "seconds";
                    default 300; // five minutes
                    description
                      "Specifies the maximum number of seconds that a
                       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 {
        uses x509c2n:cert-to-name;
        description
         "The cert-maps container is used by a TLS-based NETCONF
          server to map the NETCONF client's presented X.509
          certificate to a NETCONF username.  If no matching and
          valid cert-to-name list entry can be found, then the
          NETCONF server MUST close the connection, and MUST NOT
          accept NETCONF messages over it.";
        reference
          "RFC WWWW: NETCONF over TLS, Section 7";
      }
    }


    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 {
          key name;
          min-elements 1;
          ordered-by user;
          description
            "User-ordered list of endpoints for this NETCONF 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 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.";
          }
        }
      }
    }

  }


  <CODE ENDS>
```

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/p
 rivate-key/certificates/certificate/name
       |              +--rw client-auth
       |                 +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
       |                 +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
       |                 +--rw cert-maps
       |                    +--rw cert-to-name* [id]
       |                       +--rw id             uint32
       |                       +--rw fingerprint    x509c2n:tls-fingerpr
 int
       |                       +--rw map-type       identityref
       |                       +--rw name           string
       +--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/p
 rivate-key/certificates/certificate/name
             |        +--rw client-auth
             |           +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw cert-maps
             |              +--rw cert-to-name* [id]
             |                 +--rw id             uint32
```

```
             |                  +--rw fingerprint    x509c2n:tls-fingerpr
 int
             |                  +--rw map-type       identityref
             |                  +--rw name           string
             +--rw connection-type
             |  +--rw (connection-type)?
             |     +--:(persistent-connection)
             |     |  +--rw persistent!
             |     |     +--rw keep-alives
             |     |        +--rw max-wait?       uint16
             |     |        +--rw max-attempts?   uint8
             |     +--:(periodic-connection)
             |        +--rw periodic!
             |           +--rw reconnect-timeout?   uint16
             +--rw reconnect-strategy
                +--rw start-with?     enumeration
                +--rw max-attempts?   uint8
```

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.

```
<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>
            <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>

    <!-- 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-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>
         <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>
       </restconf-client>
     </call-home>

   </restconf-server>
```

4.5.3. YANG Model

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

```
<CODE BEGINS> file "ietf-restconf-server@2015-10-09.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
    prefix ts;
    revision-date 2015-10-09;
```

```
  }

  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 "2015-10-09" {
    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 {
          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 YYYY: 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.";
        }
        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.";
        }
      }
    }
  }


}
```

<CODE ENDS>

5. Security Considerations

This section needs to be filled in...

6. IANA Considerations

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.

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

```
name:         ietf-keychain
namespace:    urn:ietf:params:xml:ns:yang:ietf-keychain
prefix:       kc
reference:    RFC VVVV

name:         ietf-ssh-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-server
prefix:       ssvr
reference:    RFC VVVV

name:         ietf-tls-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-server
prefix:       tsvr
reference:    RFC VVVV

name:         ietf-netconf-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-netconf-server
prefix:       ncsvr
reference:    RFC VVVV

name:         ietf-restconf-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-restconf-server
prefix:       rcsvr
reference:    RFC VVVV
```

7. Other Considerations

The YANG modules define herein do not themselves support virtual routing and forwarding (VRF). It is expected that external modules will augment in VRF designations when needed.

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, 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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, March 2011.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, December 2014.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, June 2015.

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

[draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ieft-netconf-restconf-04 (work in progress), 2014.

9.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

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.

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

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