NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

K. Watsen
Juniper Networks
March 13, 2017

# Keystore Model
# draft-ietf-netconf-keystore-01

## Abstract

This document defines a YANG data module for a system-level keystore mechanism, that might be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

## Editorial Note (To be removed by RFC Editor)

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

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

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

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

- "2017-03-13" --> the publication date of this draft

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

- Appendix A. Change Log

- Appendix B. Open Issues

## Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 14, 2017.



Table of Contents

## 1. Introduction

This document defines a YANG [RFC6020] data module for a system-level keystore mechanism, which can be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

This module provides a centralized location for security sensitive data, so that the data can be then referenced by other modules. There are two types of data that are maintained by this module:

- Private keys, and any associated public certificates.
- Sets of trusted certificates.

This document extends special consideration for systems that have Trusted Protection Modules (TPMs). These systems are unique in that the TPM must be directed to generate new private keys (it is not possible to load a private key into a TPM) and it is not possible to backup/restore the TPM's private keys as configuration.

It is not required that a system has an operating system level keystore utility to implement this module.

### 1.1. Requirements Language

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 Diagram Notation

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

- Brackets "[" and "]" enclose list keys.
- Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

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

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

## 2. The Keystore Model

The keystore module defined in this section provides a configurable object having the following characteristics:

- A semi-configurable list of private keys, each with one or more associated certificates. Private keys MUST be either preinstalled (e.g., a key associated to an IDevID [Std-802.1AR-2009] certificate), be generated by request, or be loaded by request. Each private key is MAY have associated certificates, either preinstalled or configured after creation.

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

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

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

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

### 2.1. Overview

The keystore module has the following tree diagram. Please see Section 1.2 for information on how to interpret this diagram.

```
  module: ietf-keystore
      +--rw keystore
         +--rw keys
         |  +--rw key* [name]
         |     +--rw name                                    string
         |     +--rw algorithm-identifier                    identityref
         |     +--rw private-key                             union
         |     +--ro public-key                              binary
         |     +--rw certificates
         |     |  +--rw certificate* [name]
         |     |     +--rw name     string
         |     |     +--rw value?   binary
         |     +---x generate-certificate-signing-request
         |        +---w input
         |        |  +---w subject       binary
         |        |  +---w attributes?   binary
         |        +--ro output
         |           +--ro certificate-signing-request    binary
         +--rw trusted-certificates* [name]
         |  +--rw name                   string
         |  +--rw description?           string
         |  +--rw trusted-certificate* [name]
         |     +--rw name           string
         |     +--rw certificate?   binary
         +--rw trusted-host-keys* [name]
            +--rw name                string
            +--rw description?        string
            +--rw trusted-host-key* [name]
               +--rw name        string
               +--rw host-key    binary

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

## 2.2. Example Usage

The following example illustrates what a fully configured keystore 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.

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

```
     <!-- private keys and associated certificates -->
     <keys>
       <key>
         <name>ex-rsa-key</name>
         <algorithm-identifier>rsa1024</algorithm-identifier>
         <private-key>Base64-encoded RSA Private Key</private-key>
         <public-key>Base64-encoded RSA Public Key</public-key>
         <certificates>
           <certificate>
             <name>ex-rsa-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
         </certificates>
       </key>

       <key>
         <name>tls-ec-key</name>
         <algorithm-identifier>secp256r1</algorithm-identifier>
         <private-key>Base64-encoded EC Private Key</private-key>
         <public-key>Base64-encoded EC Public Key</public-key>
         <certificates>
           <certificate>
             <name>tls-ec-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
         </certificates>
       </key>

       <key>
         <name>tpm-protected-key</name>
         <algorithm-identifier>rsa2048</algorithm-identifier>
         <private-key>Base64-encoded RSA Private Key</private-key>
         <public-key>Base64-encoded RSA Public Key</public-key>
         <certificates>
           <certificate>
             <name>builtin-idevid-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
           <certificate>
             <name>my-ldevid-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
         </certificates>
       </key>
     </keys>

     <!-- trusted netconf/restconf client certificates -->
     <trusted-certificates>
```

```
      <name>explicitly-trusted-client-certs</name>
      <description>
        Specific client authentication certificates for explicitly
        trusted clients.  These are needed for client certificates
        that are not signed by a trusted CA.
      </description>
      <trusted-certificate>
        <name>George Jetson</name>
        <certificate>Base64-encoded X.509v3</certificate>
      </trusted-certificate>
    </trusted-certificates>

    <trusted-certificates>
      <name>explicitly-trusted-server-certs</name>
      <description>
        Specific server authentication certificates for explicitly
        trusted servers.  These are needed for server certificates
        that are not signed by a trusted CA.
      </description>
      <trusted-certificate>
        <name>Fred Flintstone</name>
        <certificate>Base64-encoded X.509v3</certificate>
      </trusted-certificate>
    </trusted-certificates>

    <!-- trust anchors (CA certs) for authenticating clients -->
    <trusted-certificates>
      <name>deployment-specific-ca-certs</name>
      <description>
        Trust anchors (i.e. CA certs) that are used to authenticate
        client connections.  Clients are authenticated if their
        certificate has a chain of trust to one of these configured
        CA certificates.
      </description>
      <trusted-certificate>
        <name>ca.example.com</name>
        <certificate>Base64-encoded X.509v3</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>Base64-encoded X.509v3</certificate>
      </trusted-certificate>
    </trusted-certificates>

    <!-- trusted SSH host keys -->
    <trusted-host-keys>
      <name>explicitly-trusted-ssh-host-keys</name>
      <description>
        Trusted SSH host keys used to authenticate SSH servers.
        These host keys would be analogous to those stored in
        a known_hosts file in OpenSSH.
      </description>
      <trusted-host-key>
        <name>corp-fw1</name>
        <host-key>Base64-encoded OneAsymmetricKey</host-key>
      </trusted-host-key>
    </trusted-host-keys>

  </keystore>
```

The following example illustrates the "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">
      <keystore
        xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
        <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>
        </keystore>
      </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-keystore">
         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 a "certificate-expiration" notification in XML.

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

   <notification
     xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
     <eventTime>2016-07-08T00:01:00Z</eventTime>
     <certificate-expiration
       xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
       <certificate>/ks:keystore/ks:private-keys/ks:private-key\
       /ks:certificate-chains/ks:certificate-chain/ks:certificate[3]\
       </certificate>
       <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
     </certificate-expiration>
   </notification>
```

## 2.3. YANG Module

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

```
<CODE BEGINS> file "ietf-keystore@2017-03-13.yang"

module ietf-keystore {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keystore";
  prefix "ks";

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 6536: Network Configuration Protocol (NETCONF) Access
       Control Model";
  }

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

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

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


  description
   "This module defines a keystore 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 "2017-03-13" {
    description
     "Initial version";
    reference
     "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                Models";
  }

  // Identities

  identity key-algorithm {
    description
      "Base identity from which all key-algorithms are derived.";
  }

  identity rsa1024 {
    base key-algorithm;
    description
      "The RSA algorithm using a 1024-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa2048 {
    base key-algorithm;
    description
```

```
      "The RSA algorithm using a 2048-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa3072 {
    base key-algorithm;
    description
      "The RSA algorithm using a 3072-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa4096 {
    base key-algorithm;
    description
      "The RSA algorithm using a 4096-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa7680 {
    base key-algorithm;
    description
      "The RSA algorithm using a 7680-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa15360 {
    base key-algorithm;
    description
      "The RSA algorithm using a 15360-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity secp192r1 {
    base key-algorithm;
    description
      "The secp192r1 algorithm.";
    reference
      "RFC5480:
```

```
        Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp256r1 {
    base key-algorithm;
    description
      "The secp256r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp384r1 {
    base key-algorithm;
    description
      "The secp384r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp521r1 {
    base key-algorithm;
    description
      "The secp521r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  // data model

  container keystore {
    nacm:default-deny-write;
    description
      "The keystore contains both active material (e.g., private keys
       and passwords) and passive material (e.g., trust anchors).

       The active material can be used to support either a server (e.g.,
       a TLS/SSH server's private) or a client (a private key used for
       TLS/SSH client-certificate based authentication, or a password
       used for SSH/HTTP-client authentication).

       The passive material can be used to support either a server
       (e.g., client certificates to trust) or clients (e.g., server
       certificates to trust).";

    container keys {
```

```
      description
        "A list of keys maintained by the keystore.";
      list key {
        key name;
        description
          "A key maintained by the keystore.";
        leaf name {
          type string;
          description
            "An arbitrary name for the key.";
        }
        leaf algorithm-identifier {
          type identityref {
            base "key-algorithm";
          }
          mandatory true;
          description
            "Identifies which algorithm is to be used with the key.
             This value determines how the 'private-key' and 'public-
             key' fields are interpreted.";
             // no params, such as in RFC 5912? (no are set for algs
             // we care about, but what about the future?
        }
        leaf private-key {
          nacm:default-deny-all;
          type union {
            type binary;
            type enumeration {
              enum "RESTRICTED" {
                description
                 "The private key is restricted due to access-control.";
              }
              enum "INACCESSIBLE" {
                description
                 "The private key is inaccessible due to being protected
                  by the cryptographic hardware modules (e.g., a TPM).";
              }
            }
          }
          mandatory true;
          description
            "A binary string that contains the value of the private
             key. The interpretation of the content is defined in the
             registration of the key algorithm.  For example, a DSA key
             is an INTEGER, an RSA key is represented as RSAPrivateKey
             as defined in [RFC3447], and an Elliptic Curve Cryptography
             (ECC) key is represented as ECPrivateKey as defined in
             [RFC5915]"; // text lifted from RFC5958
```

```
        }

        // no key usage (ref: RFC 5912, pg 101 -- too X.509 specific?)

        leaf public-key {
          type binary;
          config false;
          mandatory true;
          description
            "A binary string that contains the value of the public
             key. The interpretation of the content is defined in the
             registration of the key algorithm.  For example, a DSA key
             is an INTEGER, an RSA key is represented as RSAPublicKey
             as defined in [RFC3447], and an Elliptic Curve Cryptography
             (ECC) key is represented using the 'publicKey' described in
             [RFC5915]";
        }
        container certificates {
          description
            "Certificates associated with this private key.  More
             than one certificate per key is enabled to support,
             for instance, a TPM-protected key that has associated
             both IDevID and LDevID certificates.";
          list certificate {
            key name;
            description
              "A certificate for this private key.";
            leaf name {
              type string;
              description
                "An arbitrary name for the certificate.  The name
                 must be a unique across all keys, not just within
                 this key.";
            }
            leaf value {
              type binary;
              description
               "An unsigned PKCS #7 SignedData structure, as specified
                by Section 9.1 in RFC 2315, containing just certificates
                (no content, signatures, or CRLs), encoded using ASN.1
                distinguished encoding rules (DER), as specified in
                ITU-T X.690.

                This structure contains, in order, the certificate
                itself and all intermediate certificates leading up
                to a trust anchor certificate.  The certificate MAY
                optionally include the trust anchor certificate.";
              reference
```

```
                "RFC 2315:
                   PKCS #7: Cryptographic Message Syntax Version 1.5.
                 ITU-T X.690:
                   Information technology - ASN.1 encoding rules:
                   Specification of Basic Encoding Rules (BER),
                   Canonical Encoding Rules (CER) and Distinguished
                   Encoding Rules (DER).";
            }
          }
        }
        action generate-certificate-signing-request {
          description
            "Generates a certificate signing request structure for
             the associated private key using the passed subject and
             attribute values.  Please review both the Security
             Considerations and Design Considerations sections in
             RFC VVVV for more information regarding this action
             statement.";
          input {
            leaf subject {
              type binary;
              mandatory true;
              description
                "The 'subject' field from the CertificationRequestInfo
                 structure as specified by RFC 2986, Section 4.1 encoded
                 using the ASN.1 distinguished encoding rules (DER), as
                 specified in ITU-T X.690.";
              reference
                "RFC 2986:
                   PKCS #10: Certification Request Syntax Specification
                   Version 1.7.
                 ITU-T X.690:
                    Information technology - ASN.1 encoding rules:
                    Specification of Basic Encoding Rules (BER),
                    Canonical Encoding Rules (CER) and Distinguished
                    Encoding Rules (DER).";
            }
            leaf attributes {
              type binary;
              description
               "The 'attributes' field from the CertificationRequestInfo
                structure as specified by RFC 2986, Section 4.1 encoded
                using the ASN.1 distinguished encoding rules (DER), as
                specified in ITU-T X.690.";
              reference
                "RFC 2986:
                   PKCS #10: Certification Request Syntax Specification
                   Version 1.7.
```

```
                ITU-T X.690:
                   Information technology - ASN.1 encoding rules:
                   Specification of Basic Encoding Rules (BER),
                   Canonical Encoding Rules (CER) and Distinguished
                   Encoding Rules (DER).";
            }
          }
          output {
            leaf certificate-signing-request {
              type binary;
              mandatory true;
              description
                "A CertificationRequest structure as specified by RFC
                 2986, Section 4.1 encoded using the ASN.1 distinguished
                 encoding rules (DER), as specified in ITU-T X.690.";
              reference
                "RFC 2986:
                   PKCS #10: Certification Request Syntax Specification
                   Version 1.7.
                 ITU-T X.690:
                   Information technology - ASN.1 encoding rules:
                   Specification of Basic Encoding Rules (BER),
                   Canonical Encoding Rules (CER) and Distinguished
                   Encoding Rules (DER).";

            }
          }
        }
      }
    }

    list trusted-certificates {
      key name;
      description
        "A list of trusted certificates.  These certificates
         can be used by a server to authenticate clients, or by
         clients to authenticate servers.  The certificates may
         be endpoint specific or for certificate authorities,
         to authenticate many clients at once.  Each list of
         certificates SHOULD be specific to a purpose, as the
         list as a whole may be referenced by other modules.
         For instance, a NETCONF server model might point to
         a list of certificates to use when authenticating
         client 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 trusted certificate for a specific use.  Note, this
           'certificate' is a list in order to encode any
           associated intermediate certificates.";
        leaf name {
          type string;
          description
            "An arbitrary name for this trusted certificate. Must
             be unique across all lists of trusted certificates
             (not just this list) so that a leafref to it from
             another module can resolve to unique values.";
        }
        leaf certificate {  // rename to 'data'?
          type binary;
          description
            "An X.509 v3 certificate structure as specified by RFC
             5280, Section 4 encoded using the ASN.1 distinguished
             encoding rules (DER), as specified in ITU-T X.690.";
          reference
            "RFC 5280:
               Internet X.509 Public Key Infrastructure Certificate
               and Certificate Revocation List (CRL) Profile.
             ITU-T X.690:
                Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
                Encoding Rules (DER).";
        }
      }
    }

    list trusted-host-keys {
      key name;
      description
        "A list of trusted host-keys.  These host-keys can be used
         by clients to authenticate SSH servers.  The host-keys are
         endpoint specific.  Each list of host-keys SHOULD be
         specific to a purpose, as the list as a whole may be
         referenced by other modules.  For instance, a NETCONF
```

```
         client model might point to a list of host-keys to use
         when authenticating servers host-keys.";
      leaf name {
        type string;
        description
          "An arbitrary name for this list of trusted SSH host keys.";
      }
      leaf description {
        type string;
        description
          "An arbitrary description for this list of trusted SSH host
           keys.";
      }
      list trusted-host-key {
        key name;
        description
          "A trusted host key.";
        leaf name {
          type string;
          description
            "An arbitrary name for this trusted host-key. Must be
             unique across all lists of trusted host-keys (not just
             this list) so that a leafref to it from another module
             can resolve to unique values.

             Note that, for when the SSH client is able to listen
             for call-home connections as well, there is no reference
             identifier (e.g., hostname, IP address, etc.) that it
             can use to uniquely identify the server with.  The
             call-home draft recommends SSH servers use X.509v3
             certificates (RFC6187) when calling home.";
        }
        leaf host-key {  // rename to 'data'?
          type binary;
          mandatory true;
          description  // is this the correct type?
            "An OneAsymmetricKey 'publicKey' structure as specified
             by RFC 5958, Section 2 encoded using the ASN.1
             distinguished encoding rules (DER), as specified
             in ITU-T X.690.";
          reference
            "RFC 5958:
                Asymmetric Key Packages
             ITU-T X.690:
                Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
                Encoding Rules (DER).";
```

```
        }
      }
    }
  }

  notification certificate-expiration {
    description
      "A notification indicating that a configured certificate is
       either about to expire or has already expired.  When to send
       notifications is an implementation specific decision, but
       it is RECOMMENDED that a notification be sent once a month
       for 3 months, then once a week for four weeks, and then once
       a day thereafter.";
    leaf certificate {
      type instance-identifier;
      mandatory true;
      description
        "Identifies which certificate is expiring or is expired.";
    }
    leaf expiration-date {
      type yang:date-and-time;
      mandatory true;
      description
        "Identifies the expiration date on the certificate.";
    }
  }

}
```

<CODE ENDS>

## 3. Design Considerations

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

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

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

## 4. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

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

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. This being the case, the top-level node in this module is marked with the NACM value 'default-deny-write'.

/keystore/keys/key/private-key: When writing this node, implementations MUST ensure that the strength of the key being configured is not greater than the strength of the underlying secure transport connection over which it is communicated. Implementations SHOULD fail the write-request if ever the strength of the private key is greater then the strength of the underlying transport, and alert the client that the strength of the key may have been compromised. Additionally, when deleting this node, implementations SHOULD automatically (without explicit request) zeroize these keys in the most secure manner available, so as to prevent the remnants of their persisted storage locations from being analyzed in any meaningful way.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or

notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/keystore/keys/key/private-key: This node is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The best reason for returning this node is to support backup/restore type workflows. This being the case, this node is marked with the NACM value 'default-deny-all'.

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

generate-certificate-signing-request: For this RPC operation, it is RECOMMENDED that implementations assert channel binding [RFC5056], so as to ensure that the application layer that sent the request is the same as the device authenticated in the secure transport layer was established.

## 5. IANA Considerations

### 5.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

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

### 5.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registration is requested:

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

## 6. 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, Juergen Schoenwaelder; Phil Shafer, Sean Turner, and Bert Wijnen.

## 7. References

### 7.1. Normative References

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

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <http://www.rfc-editor.org/info/rfc2986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

[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>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

### 7.2. Informative References

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

[RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <http://www.rfc-editor.org/info/rfc4211>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <http://www.rfc-editor.org/info/rfc5056>.

[RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <http://www.rfc-editor.org/info/rfc5914>.

[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>.

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

[Std-802.1AR-2009] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

## Appendix A. Change Log

### A.1. server-model-09 to 00

- This draft was split out from draft-ietf-netconf-server-model-09.
- Removed key-usage parameter from generate-private-key action.
- Now /private-keys/private-key/certificates/certificate/name must be globally unique (unique across all private keys).
- Added top-level 'trusted-ssh-host-keys' and 'user-auth-credentials' to support SSH client modules.

### A.2. keychain-00 to keystore-00

- Renamed module from "keychain" to "keystore" (Issue #3)

### A.3. 00 to 01

- Replaced the 'certificate-chain' structures with PKCS#7 structures. (Issue #1)
- Added 'private-key' as a configurable data node, and removed the 'generate-private-key' and 'load-private-key' actions. (Issue #2)
- Moved 'user-auth-credentials' to the ietf-ssh-client module. (Issues #4 and #5)

## Appendix B. Open Issues

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

## Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net