Keystore Model
draft-ietf-netconf-keystore-00

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.

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

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

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

- "VVVV" --> the assigned RFC value for this draft
- "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-call-home

Artwork in this document contains placeholder values for ports pending IANA assignment from "draft-ietf-netconf-call-home". Please apply the following replacements:
- "7777" --> the assigned port value for "netconf-ch-ssh"
- "8888" --> the assigned port value for "netconf-ch-tls"
- "9999" --> the assigned port value for "restconf-ch-tls"

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

- "2016-10-31" --> 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 May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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 private-keys
            +---rw private-key* [name]
                +---rw name                         string
                +---ro algorithm?                   identityref
                +---ro key-length?                  uint32
                +---ro public-key                   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                  identityref
                |    +---w key-length?                uint32
            +---x load-private-key
                +---w input
                |    +---w name                       string
                |    +---w private-key                binary
```
2.2. Example Usage

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

POST https://example.com/restconf/data/ietf-keystore:keystore/\private-keys/load-private-key HTTP/1.1
HOST: example.com
Content-Type: application/yang.operation+xml

<input xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
  <name>ex-key-sect571r1</name>
  <private-key>
    NGcEk3UE90cnNFVjRwTUNbd0VBQWFQ0FSXdn20VCk1CMEdBMVVkRGd\VEJiZ0JTWEtdbUEKMhprRHVOVTKvVHFLNWd4cFJB212OYUUOcEERZd05ER\V6QVJCZ05WqFNVNtT1INQ0JKYzNOMVpYSONDUUUNVRBN5116UG8zRREF\Z05WSF4IVlqQmdNRj2nXSFBZ2hoNW9kSFJ3T2k4d1pYaGgKYYlhCc1pTN\QmdOVkJBWWRBbFZUTVJbd0rWnWURWUFVLRXdkbAp1R020Y0d4bE1RNdEQ\MkF6a3h9qUdv1QWtHR0dvS1UeUc1SVROw0mK3B0R2FeXVDMjBrd2kvZ\NQmdOVhkSTUJB2jhFCkFqQUNQTRHQTFZER3RUIvd1FFQxJSGdEQnBC\WmdsK2gyTTg3QmtGMjNh8NdCdFFVa4304gRyBRTFwSt4ZVRJbVFVM\lLQl1sdWpOcjFTMnRLR05EMUC2OJPk2FWNGw2NTdZNCtaddVJMGpRZYj\zSNwSdSwXBCyN4dmtNanFtZjJma3RqZHBxeFppUUTbndZWTFZ2wot\2SPZnpZNEhONApXy0pTaUpZ2xtYs3RTORU2ZS9RdGp4NU1XZmdvN2\WpiMJB2rIhoaG-YQnNaUzVqY215aU9L=
  </private-key>
</input>

RESPONSE
--------

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

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">
    </keystore>
  </action>
</rpc>
<private-keys>
  <private-key>
    <name>ex-key-sect571r1</name>
    <generate-certificate-signing-request>
      <subject>
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2R
        manZvO3NKzMpdmhzZGZpbHVidjtvc21kZmhidm1lbHNlM0
        Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmR6Zgo=
      </subject>
      <attributes>
        bwtakWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvut4
        arnZvO3NKzMpdmhzZGZpbHVidjtvc21kZmhidm1lbHNkYm
        Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmC6Rhp=
      </attributes>
    </generate-certificate-signing-request>
  </private-key>
</private-keys>

Watsen & Wu                Expires May 4, 2017                  [Page 9]
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 have been configured.

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

<!-- private keys and associated certificates -->
<private-keys>
  <private-key>
    <name>my-rsa-user-key</name>
    <algorithm>rsa</algorithm>
    <public-key>
      cztvaWRoc2RmZ2tqaHNKZmdramRzZnZzGtmam5idnV02RmanZvO3NkZmJpdmhzZGzpHVidjtvc21kZmhidml1bHNKymZ2aXNiZGzpYmhzZG87ZmJvO3NkZ25iO29pLmRZgo=
    </public-key>
  </private-key>

  <private-key>
    <name>my-ec-user-key</name>
    <algorithm>secp256r1</algorithm>
    <public-key>
      mJpdmhzZGzpHVidjtvc21kZmhidml1bHNKymZ2aXNiZGzpYmhzZG87Zm
    </public-key>
  </private-key>
</private-keys>

</keystore>
```
<public-key>

<certificate-chains>

<certificate-chain>
  <name>my-ec-chain</name>
  <certificate>
    0F3SUJBZ0lKQUpRT2t3bGpNK2pjtUEwR0NTcUdTSWIzRFFQkJRvUZKYo2ZWURiR0lPNDB4aj1Pb3JtREdsRUNCVTFvVG1rTmmpBME1Rc3diR1V4RXpBUkJnT1ZQ4UIU2t0OUIRDQkpjm04w1hJd2daOHdEU1
    LSotLS1CUDdJTiBRVJUSUZQJ0FURS0tLS0tCk1JSUNrekNDWQz5ZK529aWh2Y04KQVFQkJRQRnWTBBTU1HSkFvR0JBTXvVzmFNPENv3FNRRF4Q3pSBkJnT1YIKQzFZVEFsVlRNukF3RDzRF2RUUfF2rS2Ud
    GdGNYeGxGuNuk13RVEzRFZRUURFd3BEvWt3Z1NVTnpkV1Z5UT10WApEmKKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhStUJBZjhFQ0
    RB0FRSC9BZ0VByTUEwR0NTcUdTSWIzRFFQpCUVvBQRHvKfmMnx
    rWmFGNWcyAR6MVihzN2ZPbNcNAeG00SHRhBStadbHPlzF1338x
    Txp4YJKcBpDSH1LcKl1vc9GVZvRtV1RvS1VDeETE40NEY2Zmk2d
    c4d0tSEElkYW1WL0pGtm1Q50XSTF4K11aDzmazcrQzQ1QXg1RWV
    SWMz2TgotLSotLUVORCBDRVJUSUZQJ0FURS0tLS0tCg==
  </certificate>
</certificate-chain>

<certificate-chains>

<certificate-chain>
  <name>default-idevid-chain</name>
  <certificate>
    0F3SUJBZ0lKQUpRT2t3bGpNK2pjtUEwR0NTcUdTSWIzRFFQkJRvUZKYo2ZWURiR0lPNDB4aj1Pb3JtREdsRUNCVTFvVG1rTmmpBME1Rc3diR1V4RXpBUkJnT1ZQ4UIU2t0OUIRDQkpjm04w1hJd2daOHdEU1
    LSotLS1CUDdJTiBRVJUSUZQJ0FURS0tLS0tCk1JSUNrekNDWQz5ZK529aWh2Y04KQVFQkJRQRnWTBBTU1HSkFvR0JBTXvVzmFNPENv3FNRRF4Q3pSBkJnT1YIKQzFZVEFsVlRNukF3RDzRF2RUUfF2rS2Ud
    GdGNYeGxGuNuk13RVEzRFZRUURFd3BEvWt3Z1NVTnpkV1Z5UT10WApEmKKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhStUJBZjhFQ0
    RB0FRSC9BZ0VByTUEwR0NTcUdTSWIzRFFQpCUVvBQRHvKfmMnx
    rWmFGNWcyAR6MVihzN2ZPbNcNAeG00SHRhBStadbHPlzF1338x
    Txp4YJKcBpDSH1LcKl1vc9GVZvRtV1RvS1VDeETE40NEY2Zmk2d
    c4d0tSEElkYW1WL0pGtm1Q50XSTF4K11aDzmazcrQzQ1QXg1RWV
  </certificate>
</certificate-chain>

<private-key>

<name>tpm-protected-key</name>
<algorithm>sect571r1</algorithm>

<public-key>
    cztvaWR0c2RmZ2tgaNKZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
    mJpdmhzzGzpbHVidjtvc21kZmhidml1bHNkYmZ2aXNiZGZpYmhzZG87zm
    Jvo3NkZ25iO29pLmR6Zgo=
</public-key>

<private-key>

<name>default-idevid-chain</name>
<certificate>
    0F3SUJBZ0lKQUpRT2t3bGpNK2pjtUEwR0NTcUdTSWIzRFFQkJRvUZKYo2ZWURiR0lPNDB4aj1Pb3JtREdsRUNCVTFvVG1rTmmpBME1Rc3diR1V4RXpBUkJnT1ZQ4UIU2t0OUIRDQkpjm04w1hJd2daOHdEU1
    LSotLS1CUDdJTiBRVJUSUZQJ0FURS0tLS0tCk1JSUNrekNDWQz5ZK529aWh2Y04KQVFQkJRQRnWTBBTU1HSkFvR0JBTXvVzmFNPENv3FNRRF4Q3pSBkJnT1YIKQzFZVEFsVlRNukF3RDzRF2RUUfF2rS2Ud
    GdGNYeGxGuNuk13RVEzRFZRUURFd3BEvWt3Z1NVTnpkV1Z5UT10WApEmKKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhStUJBZjhFQ0
    RB0FRSC9BZ0VByTUEwR0NTcUdTSWIzRFFQpCUVvBQRHvKfmMnx
    rWmFGNWcyAR6MVihzN2ZPbNcNAeG00SHRhBStadbHPlzF1338x
    Txp4YJKcBpDSH1LcKl1vc9GVZvRtV1RvS1VDeETE40NEY2Zmk2d
    c4d0tSEElkYW1WL0pGtm1Q50XSTF4K11aDzmazcrQzQ1QXg1RWV
Internet-Draft  Keystore Model  October 2016

SWM2xZWtgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
</certificate>

<certificate>
KS29aSWh2Y04KQVFFQkJRQURnWTBBTU1HskFvR0JBTXVv2mFNPE3
E11QWMnQ1RsTkNmc0d6cEw10m5ydXz0sOFR1cUJTdGZQY3N0zk1KT1
FaNznN1NWVdsM1dzaHE1bUViCkJNNitGNzdjgTAvU25FcfE0tNv
bXBDT2YKQWdNQKFZBF2pnyXz32Frd0RwURWUjBPqkJ2RUZKY1o2W
LS0tLS1C0djdTiBDVCJVUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
0F3SUJB201KUpRT2t3BpNk2pjTUEwR0NTcUdTSWizRFQQkJRJVU
FNRFF4Q3pBSkJnT1YQFQKFZVEFsvNLNUkF3RdZRFZRUUtFd2RsZUd
GdGNHeGxNu1k3RVEFZRFZRUURF3dBEWv3Z1IlNYmpkV1Z5TU1OWApE
diR1V4RxPbUKnT1ZCQ1UQ2tOU1RDQkpjM04xWhJd2aOHDuEUv1
URI10PNPB4a41jP3jREdsRUNCVTNRF1HFQFVZApdmJ1TU1QURBr0FRC192Z0BVEWv3RONTcUdTSWizRFQQpCUVVBQTRHqKFMmx
rwMwFGNwcyAF6MNvhnZ2PbneHA4eG0OSRhhstAdlPazF13Sx
</certificate>
<certificate-chain>
</certificate-chain>
</name>my-ldevid-chain</name>
</certificate>
0F3SUJB201KUpRT2t3BpNk2pjTUEwR0NTcUdTSWizRFQQkJRJVU
FNRFF4Q3pBSkJnT1YQFQKFZVEFsvNLNUkF3RdZRFZRUUtFd2RsZUd
GdGNHeGxNu1k3RVEFZRFZRUURF3dBEWv3Z1IlNYmpkV1Z5TU1OWApE
diR1V4RxPbUKnT1ZCQ1UQ2tOU1RDQkpjM04xWhJd2aOHDuEUv1
URI10PNPB4a41jP3jREdsRUNCVTNRF1HFQFVZApdmJ1TU1QURBr0FRC192Z0BVEWv3RONTcUdTSWizRFQQpCUVVBQTRHqKFMmx
rwMwFGNwcyAF6MNvhnZ2PbneHA4eG0OSRhhstAdlPazF13Sx
<name>my-ldevid-chain</name>
</certificate>
SWM2xZWtgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
</certificate>
<certificate>
KS29aSWh2Y04KQVFFQkJRQURnWTBBTU1HskFvR0JBTXVv2mFNPE3
E11QWMnQ1RsTkNmc0d6cEw10m5ydXz0sOFR1cUJTdGZQY3N0zk1KT1
FaNznN1NWVdsM1dzaHE1bUViCkJNNitGNzdjgTAvU25FcfE0tNv
bXBDT2YKQWdNQKFZBF2pnyXz32Frd0RwURWUjBPqkJ2RUZKY1o2W
LS0tLS1C0djdTiBDVCJVUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
0F3SUJB201KUpRT2t3BpNk2pjTUEwR0NTcUdTSWizRFQQkJRJVU
FNRFF4Q3pBSkJnT1YQFQKFZVEFsvNLNUkF3RdZRFZRUUtFd2RsZUd
GdGNHeGxNu1k3RVEFZRFZRUURF3dBEWv3Z1IlNYmpkV1Z5TU1OWApE
diR1V4RxPbUKnT1ZCQ1UQ2tOU1RDQkpjM04xWhJd2aOHDuEUv1
URI10PNPB4a41jP3jREdsRUNCVTNRF1HFQFVZApdmJ1TU1QURBr0FRC192Z0BVEWv3RONTcUdTSWizRFQQpCUVVBQTRHqKFMmx
rwMwFGNwcyAF6MNvhnZ2PbneHA4eG0OSRhhstAdlPazF13Sx
</certificate>
<certificate-chain>
</certificate-chain>
</name>my-ldevid-chain</name>
</certificate>
0F3SUJB201KUpRT2t3BpNk2pjTUEwR0NTcUdTSWizRFQQkJRJVU
FNRFF4Q3pBSkJnT1YQFQKFZVEFsvNLNUkF3RdZRFZRUUtFd2RsZUd
GdGNHeGxNu1k3RVEFZRFZRUURF3dBEWv3Z1IlNYmpkV1Z5TU1OWApE
diR1V4RxPbUKnT1ZCQ1UQ2tOU1RDQkpjM04xWhJd2aOHDuEUv1
URI10PNPB4a41jP3jREdsRUNCVTNRF1HFQFVZApdmJ1TU1QURBr0FRC192Z0BVEWv3RONTcUdTSWizRFQQpCUVVBQTRHqKFMmx
rwMwFGNwcyAF6MNvhnZ2PbneHA4eG0OSRhhstAdlPazF13Sx
<name>my-ldevid-chain</name>
</certificate>
SWM2xZWtgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
</certificate>

Watsen & Wu  Expires May 4, 2017  [Page 12]
<!-- trusted netconf/restconf client certificates -->

<trusted-certificates>
  <name>explicitly-trusted-client-certs</name>
  <description>
  Specific client authentication certificates that are already trusted netconf/restconf clients. These are needed for client certificates not signed by our CA.
  </description>
  <trusted-certificate>
    <name>George Jetson</name>
    <certificate>
      QmdOVkjbWVBRrbFZUTVJBd0RnWURWUFLRxdkbAp1R0Z0Y0d4bE1RNHzEq
      MkF6a3hqUD1vQWhHR0dvSIU1eUC1SvR0Wm0vK3BoR2F1eXVDMjBr2dkvZ
      25P2n2KeONApXY0pTa0pZkn2xtYW33RTRORUZKZ5R9dGp4NULXmdN2
      RV0JN2c2zKZf2SNfHeUFVkpwSmY0WtXbU00NEo5akJrQmdOVkktTTUVY
      VEJiZ0JTw6d1bUERMnhpRHVTQvKVHFLNwd4cFJZ120YUU00cER2Z05ER
      UxNQw7HQTFRQvJOtUNWV14URBTOJn1ZCQW9QjJWNApZVzF3YkdVeeE
      V6QvCJCZ25QkFNFvNvT1unjQ0JYkzNOMvYsQDUUNVRHBNs116G8RZEF
      NQmdOVkhtSUJ2JhFcfQfQUFNQTHQFvZER3UvdiFFQXsdGJeBCn
      Z05Wsf41RvQmdNRj2nSXFB2zhoNW9kSFJt3tk4dlpYaqK7TuC1pBN
      Wpi1mJb2w4oaaGyGyNaUqV215aU9LUTJNRF43pBskJn1ZCQV1UQX
      xWE15QDxZd21e1V1FQswFdp2Rs2udGdGHeXNUk13RvF2RzRUURF3b
      EVt531UYyknV21sUEwR0NTcUdSuiWRFFQkJRUFNBSedCkFC3KBC
      Wmds2kgyTT7GG3ntGMVhWbW4cDEVAwc3OEgrRkYyTFRfSr42VRJbVFFM
      TQzczfZSjk0M1FQlVzv5eUGKN2QxmXkcvC0dxUjUrbEl5NO1Y1L2Ika2M4al
      zSNwWsdWxZBCyn4dmtNanFzj3maRqZBxeFppUuTbndWZTF2Zwot
      LS0tLUorCDBRvJUSUZQ0FURS0tLS0tCg==
    </certificate>
  </trusted-certificate>
  <trusted-certificate>
    <name>Fred Flintstone</name>
    <certificate>
      V1E1FRREV3V9ZWEJ32VRDQm56QU5CZ2txaGtpRzl3MEJBUUVQGUPFOm
      pRXdnWwTDCmd2RE1RzRFSWZaSlp2bdLXW44eUhyM2h0bUFRAhUvVzV
Watsen & Wu                Expires May 4, 2017                 [Page 14]
<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>
NGcEk3UE90cnNFVjrTWUNb0VBQWFPOQ0FSSXdndz20VPck1CMEdBMVvRgdVEJiZ0JTWEdbUEKMhnpRHVTvkvVHFLNwd4cFJB2120Yu00cERZd05ERV6QVCZ205WqkFVNeRN1NUQOKKYxzNOMpYS0NDOUUNVRBNS116UG8zREFZ05WFSE4RVlqmdNRj2nSXFBZ2hoNW9kSFJ3T2k4d1pYaGqY1nHc1pTNQmdOVkJBWVRrEFZUTVJBd0RnWURUVFILXdkbAp1R0Z0Yd4bEI1NHeEQMkF6a3hqUD1VIPtHrdvs1UIeuc1SVR0Wm0vX3B02fieXVDmjBRd2kVZNQmdOVkJhFCkFqQUFNTR9QTFVZER3RUIvd1FFQXhJSGdEQn7BCWmdsK2gyTTg3QntGMr1hWb1dFFVaWc3OEqrRkYyRTFwdSt4ZVRJbVFMM1LQ1lsdWpOCjF7MnLR05EMuc2OVjpk2FVNGw2NTdZNc1adVJM2prYjkzSFI7wSdWwXBCYnA4dmtNaNtZjma3RqZHbxoFppUUtTbnwWZTF22wot25P2npZNEhONAPyXYoTaUpZK2xtYW53RTRORUXZS9RdGp4NUIXzmdvN2WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
</certificate>
</trusted-certificate>
</trusted-certificate>

<!-- trusted SSH host keys -->
<trusted-ssh-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>
VEJiZ0JTWEdbUEKMhnpRHVTvkvVHFLNwd4cFJB2120Yu00cERZd05ERNGcEk3UE90cnNFVjrTWUNb0VBQWFPOQ0FSSXdndz20VPck1CMEdBMVvRgdWpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
</host-key>
</trusted-host-key>
</trusted-ssh-host-keys>

<!-- user credentials and associated authentication methods -->
<user-auth-credentials>
<user-auth-credential>
<username>admin</username>
<auth-method>
</auth-method>
</user-auth-credential>
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>2016-08-08T14:53-05:00</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@2016-10-31.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";
  }

  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 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 "2016-10-31" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
    Models";
}

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

identity rsa {
  base key-algorithm;
  description
    "The RSA algorithm.";
  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.";
}

container keystore {
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 keystore.";

list private-key {
    key name;
description
"A private key."

leaf name {
    type string;
description
"An arbitrary name for the private key."
}

leaf algorithm {
    type identityref {
        base "key-algorithm";
    }
    config false;
description
"The algorithm used by the private key.";
}

leaf key-length {
    type uint32;
    config false;
description
"The key length of the private key.";
}
"The key-length used by the private key."
}
leaf public-key {
  type binary;
  config false;
  mandatory true;
  description
  "An OneAsymmetricKey 'publicKey' structure as specified by RFC 5958, Section 2 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690."
  reference
  "RFC 5958:
  Asymmetric Key Packages
  ITU-T X.690:
  Information technology - ASN.1 encoding rules:
  Specification of Basic Encoding Rules (BER),
  Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)."
}
container certificate-chains {
  description
  "Certificate chains associated with this private key. More than one chain per key is enabled to support, for instance, a TPM-protected key that has associated both IDevID and LDevID certificates."
  list certificate-chain {
    key name;
    description
    "A certificate chain for this public key."
    leaf name {
      type string;
      description
      "An arbitrary name for the certificate chain. The name must be a unique across all private keys, not just within this private key."
    }
    leaf-list certificate {
      type binary;
      ordered-by user;
      description
      "An X.509 v3 certificate structure as specified by RFC 5280, Section 4 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690. The list of certificates that run from the server certificate towards the trust anchor. The chain MAY include the trust anchor certificate itself."
      reference
"RFC 5280:
  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).";
}
}

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

ITU-T X.690:
Information technology - ASN.1 encoding rules:
Specification of Basic Encoding Rules (BER),
Canonical Encoding Rules (CER) and Distinguished
Encoding Rules (DER)."

};

};

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

action generate-private-key {
  description
  "Requests the device to generate a private key using the
  specified algorithm and key length.";
  input {
    leaf name {
      type string;
      mandatory true;
      description
      "The name this private-key should have when listed
      in /keystore/private-keys. As such, the passed
      value must not match any existing 'name' value.";
    }
    leaf algorithm {
      type identityref {
        base "key-algorithm";
      }
    }
  }
}
mandatory true;

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

action load-private-key {
  description
  "Requests the device to load a private key";
  input {
    leaf name {
      type string;
      mandatory true;
      description
      "The name this private-key should have when listed
      in /keystore/private-keys. As such, the passed
      value must not match any existing 'name' value.";
    }
    leaf private-key {
      type binary;
      mandatory true;
      description
      "An OneAsymmetricKey structure as specified by RFC
      5958, Section 2 encoded using the ASN.1 distinguished
      encoding rules (DER), as specified in ITU-T X.690.
      Note that this is the raw private with no shrouding
      to protect it. The strength of this private key
      MUST NOT be greater than the strength of the secure
      connection over which it is communicated. Devices
      SHOULD fail this request if ever that happens.";
      reference
      "RFC 5958:
       Asymmetric Key Packages
       ITU-T X.690:
       Information technology - ASN.1 encoding rules:
       Specification of Basic Encoding Rules (BER),
       Canonical Encoding Rules (CER) and Distinguished
       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
45280, 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-ssh-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
leaf host-key { // rename to 'data'?
type binary;
mandatory true;
description
"An OneAsymmetricKey 'publicKey' structure as specified by RFC 5958, Section 2 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.";
reference
"RFC 5958:
Asymmetric Key Packages
ITU-T X.690:
Information technology - ASN.1 encoding rules:
Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).";
}

.puts

Are the auth credentials truly limited to SSH? Could they be used by an HTTP client to log into an HTTP server? If truly just for SSH, maybe rename?
*/

container user-auth-credentials {
description
"A list of user authentication credentials that can be used by an SSH client to log into an SSH server, using any of the supported authentication methods (e.g., password, public key, client certificate, etc.).";
list user-auth-credential {
  key username;
description
  "The authentication credentials for a specific user.";
  leaf username {
    type string;
description
    "The username of this user. This will be the username used, for instance, to log into an SSH server.";
  }
} list auth-method {
  key priority;
description
  "A method of authenticating as this user.";
  leaf priority {
type uint8;
description
  "When multiple authentication methods in this list are supported by the server, the one with the lowest priority value will be the one that is used."
}
choice auth-type {
  description
  "The authentication type."
  leaf-list certificate {
    type leafref {
      path "/keystore/private-keys/private-key/" + "certificate-chains/certificate-chain/name";
    }
    ordered-by user;
    description
    "A list of references to certificates that can be used for user authentication. When multiple certificates in this list supported by the server, the one that comes before the others in the leaf-list will be used."
  }
  leaf-list public-key {
    type leafref {
      path "/keystore/private-keys/private-key/name";
    }
    ordered-by user;
    description
    "A list of references to public keys that can be used for user authentication. When multiple public keys in this list supported by the server, the one that comes before the others in the leaf-list will be used."
  }
  leaf ciphertext-password {
    type string;
    description
    "An ciphertext password. The method of encipherment and how that method can be determined from this string is implementation-specific."
  }
  leaf cleartext-password {
    type string;
    description
    "An cleartext password."
  }
}
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.";
    }
}

3. Design Considerations

This document, along with four other drafts, was split out from the original draft "draft-ietf-netconf-server-model". The split was made so that each draft would have better focus, and also because there was a desire to define client modules, in addition to server modules. The complete list of drafts that resulted from the split includes:

- draft-ietf-netconf-keystore
- draft-ietf-netconf-ssh-client-server
- draft-ietf-netconf-tls-client-server
- draft-ietf-netconf-netconf-client-server
- draft-ietf-netconf-restconf-client-server
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 it 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.

Both this document and Key Chain YANG Data Model [draft-ietf-rtgwg-yang-key-chain] regard a similar idea. The authors looked at this and agree that they two modules server different purposes and hence not worth merging into one document. To underscore this further, this document renamed its module from "ietf-keychain" to "ietf-keystore", to contrast it with the other document’s module "ietf-key-chain".

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

4. Security Considerations

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

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

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

The keystore module define within this document defines the "load-private-key" action enabling a device to load a client-supplied private key. This is a private key with no shrouding to protect it. The strength of this private key MUST NOT be greater than the strength of the underlying secure transport connection over which it is communicated. Devices SHOULD fail this request if ever the strength of the private key is greater then the strength of the underlying transport.

5. IANA Considerations

5.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registration is requested:

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
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

[draft-ietf-netconf-restconf]


7.2. Informative References

[draft-ietf-rtgwg-yang-key-chain]


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.

Appendix B. Open Issues

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

Authors’ Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks
EMail: garywu@cisco.com