NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 24, 2019

K. Watsen
Juniper Networks

September 20, 2018

YANG Data Model for a Centralized Keystore Mechanism
draft-ietf-netconf-keystore-06

Abstract

This document defines a YANG 1.1 module called "ietf-keystore" that enables centralized configuration of asymmetric keys and their associated certificates, and notification for when configured certificates are about to expire.

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:

o "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:

o "2018-09-20" --> the publication date of this draft

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

o Appendix A. Change Log

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG 1.1 [RFC7950] module called "ietf-keystore" that enables centralized configuration of asymmetric keys and their associated certificates, and notification for when configured certificates are about to expire.

This module also defines Six groupings designed for maximum reuse. These groupings include one for the public half of an asymmetric key, one for both the public and private halves of an asymmetric key, one for both halves of an asymmetric key and a list of associated certificates, one for an asymmetric key that may be configured locally or via a reference to an asymmetric key in the keystore, one for a trust anchor certificate and, lastly, one for an end entity certificate.

Special consideration has been given for systems that have cryptographic hardware, such as a Trusted Protection Module (TPM). These systems are unique in that the cryptographic hardware completely hides the private keys and must perform all private key operations. To support such hardware, the "private-key" can be the special value "permanently-hidden" and the actions "generate-hidden-key" and "generate-certificate-signing-request" can be used to direct these operations to the hardware .

This document in compliant with Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, to support keys and associated certificates installed during manufacturing (e.g., for a IDevID [Std-802.1AR-2009] certificate), it is expected that such data may appear only in <operational>.

While only asymmetric keys are currently supported, the module has been designed to enable other key types to be introduced in the future.

The module does not support protecting the contents of the keystore (e.g., via encryption), though it could be extended to do so in the future.

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

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. The Keystore Model

3.1. Tree Diagram

This section provides a tree diagrams [RFC8340] for the "ietf-keystore" module that presents both the protocol-accessible "keystore" as well the all the groupings intended for external usage.

```
module: ietf-keystore
  +--rw keystore
     +--rw asymmetric-keys
        +--rw asymmetric-key* [name]
           +--rw name                                      string
           +--rw algorithm?
           |       ct:key-algorithm-ref
           +--rw public-key?                               binary
           +--rw private-key?                              union
           +---x generate-hidden-key
           |  +---w input
           |     +---w algorithm    ct:key-algorithm-ref
           +---x install-hidden-key
           |  +---w input
           |     +---w algorithm      ct:key-algorithm-ref
           |     +---w public-key?    binary
           |     +---w private-key?   binary
           +--rw certificates
           |  +--rw certificate* [name]
           |     +--rw name                        string
           |     +--rw cert
           |     |       ct:end-entity-cert-cms
           |     +---n certificate-expiration
           |        +-- expiration-date?   yang:date-and-time
           +---x generate-certificate-signing-request
              +---w input
              |  +---w subject       binary
              |  +---w attributes?   binary
              +--ro output
                 +--ro certificate-signing-request    binary

  grouping local-or-keystore-asymmetric-key-grouping
    +-- (local-or-keystore)
       +--:(local) {local-keys-supported}?
       |  +-- algorithm?             ct:key-algorithm-ref
       |  +-- public-key?            binary
       |  +-- private-key?           union
```

```
          |  +---x generate-hidden-key
          |  |  +---w input
          |  |     +---w algorithm    ct:key-algorithm-ref
          |  +---x install-hidden-key
          |     +---w input
          |        +---w algorithm      ct:key-algorithm-ref
          |        +---w public-key?    binary
          |        +---w private-key?   binary
          +--:(keystore) {keystore-supported}?
             +-- reference?             ks:asymmetric-key-ref
     grouping local-or-keystore-end-entity-certificate-grouping
       +-- (local-or-keystore)
          +--:(local) {local-keys-supported}?
          |  +-- algorithm?                ct:key-algorithm-ref
          |  +-- public-key?               binary
          |  +-- private-key?              union
          |  +---x generate-hidden-key
          |  |  +---w input
          |  |     +---w algorithm    ct:key-algorithm-ref
          |  +---x install-hidden-key
          |  |  +---w input
          |  |     +---w algorithm      ct:key-algorithm-ref
          |  |     +---w public-key?    binary
          |  |     +---w private-key?   binary
          |  +-- cert                      ct:end-entity-cert-cms
          |  +---n certificate-expiration
          |     +-- expiration-date?   yang:date-and-time
          +--:(keystore) {keystore-supported}?
             +-- reference?
                     ks:asymmetric-key-certificate-ref
     grouping local-or-keystore-asymmetric-key-with-certs-grouping
       +-- (local-or-keystore)
          +--:(local) {local-keys-supported}?
          |  +-- algorithm?
          |  |       ct:key-algorithm-ref
          |  +-- public-key?                             binary
          |  +-- private-key?                            union
          |  +---x generate-hidden-key
          |  |  +---w input
          |  |     +---w algorithm    ct:key-algorithm-ref
          |  +---x install-hidden-key
          |  |  +---w input
          |  |     +---w algorithm      ct:key-algorithm-ref
          |  |     +---w public-key?    binary
          |  |     +---w private-key?   binary
          |  +-- certificates
          |  |  +-- certificate* [name]
          |  |     +-- name?                     string
```

```
          |  |     +-- cert                         ct:end-entity-cert-cms
          |  |     +---n certificate-expiration
          |  |        +-- expiration-date?   yang:date-and-time
          |  +---x generate-certificate-signing-request
          |     +---w input
          |     |  +---w subject       binary
          |     |  +---w attributes?   binary
          |     +--ro output
          |        +--ro certificate-signing-request    binary
          +--:(keystore) {keystore-supported}?
             +-- reference?
                     ks:asymmetric-key-ref
```

3.2. Example Usage

The following example illustrates what a fully configured keystore might look like in <operational>, as described by Section 5.3 in [RFC8342]. This datastore view illustrates data set by the manufacturing process alongside conventional configuration. This keystore instance has four keys, two having one associated certificate, one having two associated certificates, and one empty key.

[Note: '\' line wrapping for formatting only]


```
<keystore xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore"
          xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
          xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types"
          or:origin="or:intended">
  <asymmetric-keys>

    <asymmetric-key>
      <name>ex-rsa-key</name>
      <algorithm>ct:rsa1024</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <certificates>
        <certificate>
          <name>ex-rsa-cert</name>
          <cert>base64encodedvalue==</cert>
        </certificate>
      </certificates>
    </asymmetric-key>

    <asymmetric-key>
      <name>tls-ec-key</name>
      <algorithm>ct:secp256r1</algorithm>
```

```
         <private-key>base64encodedvalue==</private-key>
         <public-key>base64encodedvalue==</public-key>
         <certificates>
           <certificate>
             <name>tls-ec-cert</name>
             <cert>base64encodedvalue==</cert>
           </certificate>
         </certificates>
       </asymmetric-key>

       <asymmetric-key>
         <name>tpm-protected-key</name>
         <algorithm or:origin="or:system">ct:rsa2048</algorithm>
         <private-key or:origin="or:system">permanently-hidden</private\
   -key>
         <public-key or:origin="or:system">base64encodedvalue==</public\
   -key>
         <certificates>
           <certificate or:origin="or:system">
             <name>builtin-idevid-cert</name>
             <cert>base64encodedvalue==</cert>
           </certificate>
           <certificate>
             <name>my-ldevid-cert</name>
             <cert>base64encodedvalue==</cert>
           </certificate>
         </certificates>
       </asymmetric-key>

       <asymmetric-key>
         <name>empty-key</name>
       </asymmetric-key>

     </asymmetric-keys>
   </keystore>
```


The following example module has been constructed to illustrate the "local-or-keystore-asymmetric-key-grouping" grouping defined in the "ietf-keystore" module.

```
module ex-keystore-usage {
  yang-version 1.1;

  namespace "http://example.com/ns/example-keystore-usage";
  prefix "eku";

  import ietf-keystore {
    prefix ks;
    reference
      "RFC VVVV: YANG Data Model for a 'Keystore' Mechanism";
  }

  organization
   "Example Corporation";

  contact
   "Author: YANG Designer <mailto:yang.designer@example.com>";

  description
   "This module illustrates the grouping defined in the keystore
    draft called 'local-or-keystore-asymmetric-key-grouping'.";

  revision "YYYY-MM-DD" {
    description
     "Initial version";
    reference
     "RFC XXXX: YANG Data Model for a 'Keystore' Mechanism";
  }

  container keys {
    description
      "A container of keys.";
    list key {
      key name;
      leaf name {
        type string;
        description
          "An arbitrary name for this key.";
      }
      uses ks:local-or-keystore-asymmetric-key-grouping;
      description
        "A key which may be configured locally or be a reference to
         a key in the keystore.";
    }
  }
}
```

The following example illustrates what two configured keys, one local and the other remote, might look like. This example consistent with other examples above (i.e., the referenced key is in an example above).


```
<keys xmlns="http://example.com/ns/example-keystore-usage">
  <key>
    <name>locally-defined key</name>
    <algorithm
      xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
      ct:secp521r1
    </algorithm>
    <private-key>base64encodedvalue==</private-key>
    <public-key>base64encodedvalue==</public-key>
  </key>
  <key>
    <name>keystore-defined key</name>
    <reference>ex-rsa-key</reference>
  </key>
</keys>
```


3.3. YANG Module

This YANG module has normative references to [RFC8341] and [I-D.ietf-netconf-crypto-types], and an informative reference to [RFC8342].

<CODE BEGINS> file "ietf-keystore@2018-09-20.yang"

[Note: '\' line wrapping for formatting only]

```
module ietf-keystore {
  yang-version 1.1;

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

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC CCCC: Common YANG Data Types for Cryptography";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }
```

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

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

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


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

       Copyright (c) 2018 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 "2018-09-20" {
       description
        "Initial version";
       reference
        "RFC VVVV:
           YANG Data Model for a Centralized Keystore Mechanism";
     }

     // Features

     feature keystore-supported {
       description
        "The 'keystore-supported' feature indicates that the server
         supports the keystore.";
     }

     feature local-keys-supported {
       description
         "The 'local-keys-supported' feature indocates that the
          server supports locally-defined keys.";
```

```
     }

     // Typedefs

     typedef asymmetric-key-ref {
       type leafref {
         path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
              + "/ks:name";
         require-instance false;
       }
       description
         "This typedef enables modules to easily define a reference
          to an asymmetric key stored in the keystore. The require
          instance attribute is false to enable the referencing of
          asymmetric keys that exist only in <operational>.";
       reference
         "RFC 8342: Network Management Datastore Architecture (NMDA)";
     }

     typedef asymmetric-key-certificate-ref {
       type leafref {
         path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
              + "/ks:certificates/ks:certificate/ks:name";
         require-instance false;
       }
       description
         "This typedef enables modules to easily define a reference
          to a specific certificate associated with an asymmetric key
          stored in the keystore.  The require instance attribute is
          false to enable the referencing of certificates that exist
          only in <operational>.";
       reference
         "RFC 8342: Network Management Datastore Architecture (NMDA)";
     }


     // Groupings

   // MOVED TO CRYPTO TYPES DRAFT? - OKAY TO REMOVE HERE NOW?
   //
   //  grouping public-key-grouping {
   //    description
   //      "A public key.";
   //    leaf algorithm {
   //      type ct:key-algorithm-ref;
   //      description
   //        "Identifies the key's algorithm.  More specifically,
   //         this leaf specifies how the 'public-key' binary leaf
```

```
   //          is encoded.";
   //       reference
   //         "RFC CCCC: Common YANG Data Types for Cryptography";
   //     }
   //     leaf public-key {
   //       type binary;
   //       description
   //         "A binary that contains the value of the public key.  The
   //          interpretation of the content is defined by the key
   //          algorithm.  For example, a DSA key is an integer, an RSA
   //          key is represented as RSAPublicKey as defined in
   //          RFC 3447, and an Elliptic Curve Cryptography (ECC) key
   //          is represented using the 'publicKey' described in
   //          RFC 5915.";
   //       reference
   //         "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
   //                    RSA Cryptography Specifications Version 2.1.
   //          RFC 5915: Elliptic Curve Private Key Structure.";
   //     }
   //   }
   //
   //   grouping asymmetric-key-pair-grouping {
   //     description
   //       "A private/public key pair.";
   //     uses ct:public-key-grouping;
   //     leaf private-key {
   //       nacm:default-deny-all;
   //       type union {
   //         type binary;
   //         type enumeration {
   //           enum "permanently-hidden" {
   //             description
   //              "The private key is inaccessible due to being
   //               protected by the system (e.g., a cryptographic
   //               hardware module).  It is not possible to
   //               configure a permanently hidden key, as a real
   //               private key value must be set.  Permanently
   //               hidden keys cannot be archived or backed up.";
   //           }
   //         }
   //       }
   //       description
   //         "A binary that contains the value of the private key.  The
   //          interpretation of the content is defined by the key
   //          algorithm.  For example, a DSA key is an integer, an RSA
   //          key is represented as RSAPrivateKey as defined in
   //          RFC 3447, and an Elliptic Curve Cryptography (ECC) key
   //          is represented as ECPrivateKey as defined in RFC 5915.";
```

```
   //      reference
   //        "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
   //                   RSA Cryptography Specifications Version 2.1.
   //         RFC 5915: Elliptic Curve Private Key Structure.";
   //    }
   //    action generate-hidden-key {
   //      description
   //        "Requests the device to generate a hidden key using the
   //         specified asymmetric key algorithm.  This action is
   //         used to request the system the generate a key that
   //         is 'permanently-hidden', perhaps protected by a
   //         cryptographic hardware module.  The resulting
   //         asymmetric key values are considered operational
   //         state and hence present only in <operational>.";
   //      input {
   //        leaf algorithm {
   //          type ct:key-algorithm-ref;
   //          mandatory true;
   //          description
   //            "The algorithm to be used when generating the
   //             asymmetric key.";
   //          reference
   //            "RFC CCCC: Common YANG Data Types for Cryptography";
   //        }
   //      }
   //    } // end generate-hidden-key
   //    action install-hidden-key {
   //      description
   //        "Requests the device to load the specified values into
   //         a hidden key.  The resulting asymmetric key values are
   //         considered operational state and hence present only in
   //         <operational>.";
   //      input {
   //        leaf algorithm {
   //          type ct:key-algorithm-ref;
   //          mandatory true;
   //          description
   //            "The algorithm to be used when generating the
   //             asymmetric key.";
   //          reference
   //            "RFC CCCC: Common YANG Data Types for Cryptography";
   //        }
   //        leaf public-key {
   //          type binary;
   //          description
   //            "A binary that contains the value of the public key.
    //           The interpretation of the content is defined by the key
   //             algorithm.  For example, a DSA key is an integer, an
```

```
   //             RSA key is represented as RSAPublicKey as defined in
   //             RFC 3447, and an Elliptic Curve Cryptography (ECC) key
   //             is represented using the 'publicKey' described in
   //             RFC 5915.";
   //          reference
   //            "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
   //                       RSA Cryptography Specifications Version 2.1.
   //             RFC 5915: Elliptic Curve Private Key Structure.";
   //        }
   //        leaf private-key {
   //          type binary;
   //          description
   //            "A binary that contains the value of the private key.
   //             The interpretation of the content is defined by the k\
   ey
   //           algorithm.  For example, a DSA key is an integer, an RSA
   //             key is represented as RSAPrivateKey as defined in
   //             RFC 3447, and an Elliptic Curve Cryptography (ECC) key
   //            is represented as ECPrivateKey as defined in RFC 5915.\
   ";
   //          reference
   //            "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
   //                       RSA Cryptography Specifications Version 2.1.
   //             RFC 5915: Elliptic Curve Private Key Structure.";
   //        }
   //      }
   //    } // end install-hidden-key
   //  }
   //
   //  grouping trust-anchor-cert-grouping {
   //    description
   //     "A certificate, and a notification for when it might expire.";
   //    leaf cert {
   //      type ct:trust-anchor-cert-cms;
   //      mandatory true;
   //      description
   //       "The binary certificate data for this certificate.";
   //      reference
   //       "RFC YYYY: Common YANG Data Types for Cryptography";
   //    }
   //  }
   //
   //  grouping end-entity-cert-grouping {
   //    description
   //     "A certificate, and a notification for when it might expire.";
   //    leaf cert {
   //      type ct:end-entity-cert-cms;
   //      mandatory true;
```

```
   //      description
   //       "The binary certificate data for this certificate.";
   //      reference
   //        "RFC YYYY: Common YANG Data Types for Cryptography";
   //    }
   //    notification certificate-expiration {
   //      description
   //        "A notification indicating that the 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 until the issue is resolved.";
   //      leaf expiration-date {
   //        type yang:date-and-time;
   //        //mandatory true;
   //        description
   //          "Identifies the expiration date on the certificate.";
   //      }
   //    }
   //  }
   //
   //  grouping asymmetric-key-pair-with-certs-grouping {
   //    description
   //      "A private/public key pair and associated certificates.";
   //    uses ct:asymmetric-key-pair-grouping;
   //    container certificates {
   //      description
   //        "Certificates associated with this asymmetric key.
   //         More than one certificate supports, for instance,
   //         a TPM-protected asymmetric key that has both IDevID
   //         and LDevID certificates associated.";
   //      list certificate {
   //        must "../../algorithm
   //               and ../../public-key
   //                 and ../../private-key";
   //        key name;
   //        description
   //          "A certificate for this asymmetric key.";
   //        leaf name {
   //          type string;
   //          description
   //            "An arbitrary name for the certificate.";
   //        }
   //        uses ct:end-entity-cert-grouping;
   //      } // end certifcate
   //    } // end certificates
   //    action generate-certificate-signing-request {
```

```
   //      description
   //        "Generates a certificate signing request structure for
   //         the associated asymmetric key using the passed subject
   //         and attribute values.  The specified assertions need
   //         to be appropriate for the certificate's use.  For
   //         example, an entity certificate for a TLS server
   //         SHOULD have values that enable clients to satisfy
   //         RFC 6125 processing.";
   //      input {
   //        leaf subject {
   //          type binary;
   //          mandatory true;
   //          description
   //           "The 'subject' field per 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 Syntaxi
   //                         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 structure
   //             CertificationRequestInfo 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 4.2 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).";
   //
   //        }
   //      } // end output
   //    } // end generate-certificate-signing-request
   //  }
   //
   // MOVED TO CRYPTO TYPES DRAFT? - OKAY TO REMOVE HERE NOW?

     grouping local-or-keystore-asymmetric-key-grouping {
       description
         "A grouping that expands to allow the key to be either stored
          locally within the using data model, or be a reference to an
          asymmetric key stored in the keystore.";
       choice local-or-keystore {
         mandatory true;
         case local {
           if-feature "local-keys-supported";
           uses ct:asymmetric-key-pair-grouping;
         }
         case keystore {
           if-feature "keystore-supported";
           leaf reference {
             type ks:asymmetric-key-ref;
             description
               "A reference to a value that exists in the keystore.";
           }
         }
         description
           "A choice between an inlined definition and a definition
            that exists in the keystore.";
       }
     }
```

```
     grouping local-or-keystore-asymmetric-key-with-certs-grouping {
       description
         "A grouping that expands to allow the key to be either stored
          locally within the using data model, or be a reference to an
          asymmetric key stored in the keystore.";
       choice local-or-keystore {
         mandatory true;
         case local {
           if-feature "local-keys-supported";
           uses ct:asymmetric-key-pair-with-certs-grouping;
         }
         case keystore {
           if-feature "keystore-supported";
           leaf reference {
             type ks:asymmetric-key-ref;
             description
               "A reference to a value that exists in the keystore.";
           }
         }
         description
           "A choice between an inlined definition and a definition
            that exists in the keystore.";
       }
     }

     grouping local-or-keystore-end-entity-certificate-grouping {
       description
         "A grouping that expands to allow the end-entity certificate
          (and the associated private key) to be either stored locally
          within the using data model, or be a reference to a specific
          certificate in the keystore.";
       choice local-or-keystore {
         mandatory true;
         case local {
           if-feature "local-keys-supported";
           uses ct:asymmetric-key-pair-grouping;
           uses ct:end-entity-cert-grouping;
         }
         case keystore {
           if-feature "keystore-supported";
           leaf reference {
             type ks:asymmetric-key-certificate-ref;
             description
               "A reference to a value that exists in the keystore.";
           }
         }
         description
           "A choice between an inlined definition and a definition
```

```
            that exists in the keystore.";
       }
     }


     // protocol accessible nodes

     container keystore {
       nacm:default-deny-write;

       description
         "The keystore contains a list of keys.";

       container asymmetric-keys {
         description
           "A list of asymmetric keys.";
         list asymmetric-key {
           must "(algorithm and public-key and private-key)
                  or not (algorithm or public-key or private-key)";
           key name;
           description
             "An asymmetric key.";
           leaf name {
             type string;
             description
               "An arbitrary name for the asymmetric key.";
           }
           uses ct:asymmetric-key-pair-with-certs-grouping;
         } // end asymmetric-key

       } // end asymmetric-keys
     } // end keystore


   }
   <CODE ENDS>
```

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) [RFC8341] 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, etc., can dramatically alter the implemented security policy. For this reason, the NACM extension "default-deny-write" has been set for the entire data tree.

/keystore/asymmetric-keys/asymmetric-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/asymmetric-keys/asymmetric-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. For this reason, the NACM extension "default-deny-all" has been set for this data node. Note that this extension is inherited from the grouping in the [I-D.ietf-netconf-crypto-types] module.

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:       ks
reference:    RFC VVVV
```

6. References

6.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-00 (work in progress), June 2018.

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

6.2. Informative References

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

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[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. 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)

A.2. 01 to 02

- Added back 'generate-private-key' action.

- Removed 'RESTRICTED' enum from the 'private-key' leaf type.

- Fixed up a few description statements.

A.3. 02 to 03

- Changed draft's title.

- Added missing references.

- Collapsed sections and levels.

- Added RFC 8174 to Requirements Language Section.

- Renamed 'trusted-certificates' to 'pinned-certificates'.

- Changed 'public-key' from config false to config true.

- Switched 'host-key' from OneAsymmetricKey to definition from RFC 4253.

A.4. 03 to 04

- Added typedefs around leafrefs to common keystore paths

- Now tree diagrams reference ietf-netmod-yang-tree-diagrams

- Removed Design Considerations section

- Moved key and certificate definitions from data tree to groupings

A.5. 04 to 05

- Removed trust anchors (now in their own draft)
- Added back global keystore structure
- Added groupings enabling keys to either be locally defined or a reference to the keystore.

A.6. 05 to 06

- Added feature "local-keys-supported"
- Added nacm:default-deny-all and nacm:default-deny-write
- Renamed generate-asymmetric-key to generate-hidden-key
- Added an install-hidden-key action
- Moved actions inside fo the "asymmetric-key" container
- Moved some groupings to draft-ietf-netconf-crypto-types

Acknowledgements

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

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net