NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 20, 2020

K. Watsen
Watsen Networks
H. Wang
Huawei

October 18, 2019

# Common YANG Data Types for Cryptography
draft-ietf-netconf-crypto-types-11

## Abstract

This document defines YANG identities, typedefs, the groupings useful for cryptographic applications.

## 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:

- "XXXX" --> 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:

- "2019-10-18" --> the publication date of this draft

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

- Appendix B. Change Log

## 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 https://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 April 20, 2020.

Copyright Notice



Table of Contents

## 1. Introduction

This document defines a YANG 1.1 [RFC7950] module specifying identities, typedefs, and groupings useful for cryptography.

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.

## 2. The Crypto Types Module

### 2.1. Tree Diagram

This section provides a tree diagram [RFC8340] for the "ietf-crypto-types" module. Only the groupings as represented, as tree diagrams have no means to represent identities or typedefs.

```
module: ietf-crypto-types

  grouping symmetric-key-grouping
    +-- algorithm           encryption-algorithm-t
    +-- key-format?         identityref
    +-- (key-type)
       +--:(key)
       |  +-- key?          binary
       +--:(hidden-key)
```

```
        +-- hidden-key?   empty
  grouping public-key-grouping
    +-- algorithm            asymmetric-key-algorithm-t
    +-- public-key-format?   identityref
    +-- public-key           binary
  grouping asymmetric-key-pair-grouping
    +-- algorithm                   asymmetric-key-algorithm-t
    +-- public-key-format?          identityref
    +-- public-key                  binary
    +-- private-key-format?         identityref
    +-- (private-key-type)
       +--:(private-key)
```





```
       |  +-- private-key?          binary
       +--:(hidden-private-key)
          +-- hidden-key?   empty
  grouping trust-anchor-cert-grouping
    +-- cert?                    trust-anchor-cert-cms
    +---n certificate-expiration
       +-- expiration-date    yang:date-and-time
  grouping trust-anchor-certs-grouping
    +-- cert*                    trust-anchor-cert-cms
    +---n certificate-expiration
       +-- expiration-date    yang:date-and-time
  grouping end-entity-cert-grouping
    +-- cert?                    end-entity-cert-cms
    +---n certificate-expiration
       +-- expiration-date    yang:date-and-time
  grouping end-entity-certs-grouping
    +-- cert*                    end-entity-cert-cms
    +---n certificate-expiration
       +-- expiration-date    yang:date-and-time
  grouping asymmetric-key-pair-with-cert-grouping
    +-- algorithm
    |       asymmetric-key-algorithm-t
    +-- public-key-format?                      identityref
    +-- public-key                              binary
    +-- private-key-format?                     identityref
    +-- (private-key-type)
    |  +--:(private-key)
    |  |  +-- private-key?                      binary
    |  +--:(hidden-private-key)
```

```
   |     +-- hidden-private-key?               empty
   +-- cert?                                   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 asymmetric-key-pair-with-certs-grouping
  +-- algorithm
  |       asymmetric-key-algorithm-t
  +-- public-key-format?                      identityref
  +-- public-key                              binary
  +-- private-key-format?                     identityref
  +-- (private-key-type)
  |  +--:(private-key)
  |  |  +-- private-key?                      binary
```





```
   |  +--:(hidden-private-key)
   |     +-- hidden-private-key?               empty
   +-- certificates
   |  +-- certificate* [name]
   |     +-- name?                    string
   |     +-- cert?                    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
```

## 2.2. YANG Module

This module has normative references to [RFC2404], [RFC3565], [RFC3686], [RFC4106], [RFC4253], [RFC4279], [RFC4309], [RFC4494], [RFC4543], [RFC4868], [RFC5280], [RFC5652], [RFC5656], [RFC6187], [RFC6991], [RFC7919], [RFC8268], [RFC8332], [RFC8341], [RFC8422], [RFC8446], and [ITU.X690.2015].

This module has an informational reference to [RFC2986], [RFC3174], [RFC4493], [RFC5915], [RFC6125], [RFC6234], [RFC6239], [RFC6507], [RFC8017], [RFC8032], [RFC8439].

<CODE BEGINS> file "ietf-crypto-types@2019-10-18.yang"

```
module ietf-crypto-types {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-crypto-types";
  prefix ct;

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

  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:kent+ietf@watsen.net>
     Author:   Wang Haiguang <wang.haiguang.shieldlab@huawei.com>";

  description
    "This module defines common YANG types for cryptographic
     applications.

     Copyright (c) 2019 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
      (https://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC XXXX
      (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
      itself for full legal notices.;

      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 (RFC 2119)
      (RFC 8174) when, and only when, they appear in all
      capitals, as shown here.";

  revision 2019-10-18 {
    description
      "Initial version";
    reference
      "RFC XXXX: Common YANG Data Types for Cryptography";
  }

  /***************************************/
  /*   Identities for Hash Algorithms    */
  /***************************************/

  typedef hash-algorithm-t {
    type union {
      type uint16;
      type enumeration {
        enum NONE {
          value 0;
```





```
          description
            "Hash algorithm is NULL.";
        }
        enum sha1 {
          value 1;
          status obsolete;
          description
            "The SHA1 algorithm.";
```

```
        reference
          "RFC 3174: US Secure Hash Algorithms 1 (SHA1).";
      }
      enum sha-224 {
        value 2;
        description
          "The SHA-224 algorithm.";
        reference
          "RFC 6234: US Secure Hash Algorithms.";
      }
      enum sha-256 {
        value 3;
        description
          "The SHA-256 algorithm.";
        reference
          "RFC 6234: US Secure Hash Algorithms.";
      }
      enum sha-384 {
        value 4;
        description
          "The SHA-384 algorithm.";
        reference
          "RFC 6234: US Secure Hash Algorithms.";
      }
      enum sha-512 {
        value 5;
        description
          "The SHA-512 algorithm.";
        reference
          "RFC 6234: US Secure Hash Algorithms.";
      }
      enum shake-128 {
        value 6;
        description
          "The SHA3 algorithm with 128-bits output.";
        reference
          "National Institute of Standards and Technology,
          SHA-3 Standard: Permutation-Based Hash and
          Extendable-Output Functions, FIPS PUB 202, DOI
          10.6028/NIST.FIPS.202, August 2015.";
```





```
      }
```

```
        enum shake-224 {
          value 7;
          description
            "The SHA3 algorithm with 224-bits output.";
          reference
            "National Institute of Standards and Technology,
            SHA-3 Standard: Permutation-Based Hash and
            Extendable-Output Functions, FIPS PUB 202, DOI
            10.6028/NIST.FIPS.202, August 2015.";
        }
        enum shake-256 {
          value 8;
          description
            "The SHA3 algorithm with 256-bits output.";
          reference
            "National Institute of Standards and Technology,
            SHA-3 Standard: Permutation-Based Hash and
            Extendable-Output Functions, FIPS PUB 202, DOI
            10.6028/NIST.FIPS.202, August 2015.";
        }
        enum shake-384 {
          value 9;
          description
            "The SHA3 algorithm with 384-bits output.";
          reference
            "National Institute of Standards and Technology,
            SHA-3 Standard: Permutation-Based Hash and
            Extendable-Output Functions, FIPS PUB 202, DOI
            10.6028/NIST.FIPS.202, August 2015.";
        }
        enum shake-512 {
          value 10;
          description
            "The SHA3 algorithm with 384-bits output.";
          reference
            "National Institute of Standards and Technology,
            SHA-3 Standard: Permutation-Based Hash and
            Extendable-Output Functions, FIPS PUB 202, DOI
            10.6028/NIST.FIPS.202, August 2015.";
        }
      }
    }
    default "0";
    description
      "The uint16 filed shall be set by individual protocol families
       according to the hash algorithm value assigned by IANA. The
       setting is optional and by default is 0.  The enumeration
```

```
          filed is set to the selected hash algorithm.";
      }

      /***********************************************/
      /*  Identities for Asymmetric Key Algorithms   */
      /***********************************************/

      typedef asymmetric-key-algorithm-t {
        type union {
          type uint16;
          type enumeration {
            enum NONE {
              value 0;
              description
                "Asymetric key algorithm is NULL.";
            }
            enum rsa1024 {
              value 1;
              description
                "The RSA algorithm using a 1024-bit key.";
              reference
                "RFC 8017: PKCS #1: RSA Cryptography
                 Specifications Version 2.2.";
            }
            enum rsa2048 {
              value 2;
              description
                "The RSA algorithm using a 2048-bit key.";
              reference
                "RFC 8017:
                 PKCS #1: RSA Cryptography Specifications Version 2.2.";
            }
            enum rsa3072 {
              value 3;
              description
                "The RSA algorithm using a 3072-bit key.";
              reference
                "RFC 8017:
                 PKCS #1: RSA Cryptography Specifications Version 2.2.";
            }
            enum rsa4096 {
              value 4;
              description
                "The RSA algorithm using a 4096-bit key.";
              reference
                "RFC 8017:
```

```
            PKCS #1: RSA Cryptography Specifications Version 2.2.";
        }
```





```
        enum rsa7680 {
          value 5;
          description
            "The RSA algorithm using a 7680-bit key.";
          reference
            "RFC 8017:
             PKCS #1: RSA Cryptography Specifications Version 2.2.";
        }
        enum rsa15360 {
          value 6;
          description
            "The RSA algorithm using a 15360-bit key.";
          reference
            "RFC 8017:
             PKCS #1: RSA Cryptography Specifications Version 2.2.";
        }
        enum secp192r1 {
          value 7;
          description
            "The asymmetric algorithm using a NIST P192 Curve.";
          reference
            "RFC 6090:
               Fundamental Elliptic Curve Cryptography Algorithms.
             RFC 5480:
                Elliptic Curve Cryptography Subject Public Key
                Information.";
        }
        enum secp224r1 {
          value 8;
          description
            "The asymmetric algorithm using a NIST P224 Curve.";
          reference
            "RFC 6090:
               Fundamental Elliptic Curve Cryptography Algorithms.
             RFC 5480:
               Elliptic Curve Cryptography Subject Public Key
               Information.";
        }
        enum secp256r1 {
```

```
        value 9;
        description
          "The asymmetric algorithm using a NIST P256 Curve.";
        reference
          "RFC 6090:
             Fundamental Elliptic Curve Cryptography Algorithms.
           RFC 5480:
             Elliptic Curve Cryptography Subject Public Key
             Information.";
```





```
      }
      enum secp384r1 {
        value 10;
        description
          "The asymmetric algorithm using a NIST P384 Curve.";
        reference
          "RFC 6090:
             Fundamental Elliptic Curve Cryptography Algorithms.
           RFC 5480:
             Elliptic Curve Cryptography Subject Public Key
             Information.";
      }
      enum secp521r1 {
        value 11;
        description
          "The asymmetric algorithm using a NIST P521 Curve.";
        reference
          "RFC 6090:
             Fundamental Elliptic Curve Cryptography Algorithms.
           RFC 5480:
             Elliptic Curve Cryptography Subject Public Key
             Information.";
      }
      enum x25519 {
        value 12;
        description
          "The asymmetric algorithm using a x.25519 Curve.";
        reference
          "RFC 7748:
             Elliptic Curves for Security.";
      }
      enum x448 {
```

```
            value 13;
            description
              "The asymmetric algorithm using a x.448 Curve.";
            reference
              "RFC 7748:
                 Elliptic Curves for Security.";
          }
        }
      }
      default "0";
      description
        "The uint16 filed shall be set by individual protocol
         families according to the asymmetric key algorithm value
         assigned by IANA. The setting is optional and by default
         is 0.  The enumeration filed is set to the selected
         asymmetric key algorithm.";
```





```
    }

    /*************************************/
    /*   Identities for MAC Algorithms   */
    /*************************************/

    typedef mac-algorithm-t {
      type union {
        type uint16;
        type enumeration {
          enum NONE {
            value 0;
            description
              "mac algorithm is NULL.";
          }
          enum hmac-sha1 {
            value 1;
            description
              "Generating MAC using SHA1 hash function";
            reference
              "RFC 3174: US Secure Hash Algorithm 1 (SHA1)";
          }
          enum hmac-sha1-96 {
            value 2;
            description
```

```
        "Generating MAC using SHA1 hash function";
      reference
        "RFC 2404: The Use of HMAC-SHA-1-96 within ESP and AH";
    }
    enum hmac-sha2-224 {
      value 3;
      description
        "Generating MAC using SHA2 hash function";
      reference
        "RFC 6234: US Secure Hash Algorithms
         (SHA and SHA-based HMAC and HKDF)";
    }
    enum hmac-sha2-256 {
      value 4;
      description
        "Generating MAC using SHA2 hash function";
      reference
        "RFC 6234: US Secure Hash Algorithms
         (SHA and SHA-based HMAC and HKDF)";
    }
    enum hmac-sha2-256-128 {
      value 5;
      description
```





```
        "Generating a 256 bits MAC using SHA2 hash function and
         truncate it to 128 bits";
      reference
        "RFC 4868: Using HMAC-SHA-256, HMAC-SHA-384,
         and HMAC-SHA-512 with IPsec";
    }
    enum hmac-sha2-384 {
      value 6;
      description
        "Generating a 384 bits MAC using SHA2 hash function";
      reference
        "RFC 6234: US Secure Hash Algorithms
         (SHA and SHA-based HMAC and HKDF)";
    }
    enum hmac-sha2-384-192 {
      value 7;
      description
        "Generating a 384 bits MAC using SHA2 hash function and
```

```
          truncate it to 192 bits";
      reference
        "RFC 4868: Using HMAC-SHA-256, HMAC-SHA-384,
         and HMAC-SHA-512 with IPsec";
    }
    enum hmac-sha2-512 {
      value 8;
      description
        "Generating a 512 bits MAC using SHA2 hash function";
      reference
        "RFC 6234: US Secure Hash Algorithms
         (SHA and SHA-based HMAC and HKDF)";
    }
    enum hmac-sha2-512-256 {
      value 9;
      description
        "Generating a 512 bits MAC using SHA2 hash function and
         truncate it to 256 bits";
      reference
        "RFC 4868: Using HMAC-SHA-256, HMAC-SHA-384,
         and HMAC-SHA-512 with IPsec";
    }
    enum aes-128-gmac {
      value 10;
      description
        "Generating 128-bit MAC using the Advanced Encryption
         Standard (AES) Galois Message Authentication Code
         (GMAC) as a mechanism to provide data origin
         authentication.";
      reference
```





```
        "RFC 4543:
           The Use of Galois Message Authentication Code (GMAC)
           in IPsec ESP and AH";
    }
    enum aes-192-gmac {
      value 11;
      description
        "Generating 192-bit MAC using the Advanced Encryption
         Standard (AES) Galois Message Authentication Code
         (GMAC) as a mechanism to provide data origin
         authentication.";
```

```
      reference
        "RFC 4543:
           The Use of Galois Message Authentication Code (GMAC)
           in IPsec ESP and AH";
    }
    enum aes-256-gmac {
      value 12;
      description
        "Generating 256-bit MAC using the Advanced Encryption
         Standard (AES) Galois Message Authentication Code
         (GMAC) as a mechanism to provide data origin
         authentication.";
      reference
        "RFC 4543:
           The Use of Galois Message Authentication Code (GMAC)
           in IPsec ESP and AH";
    }
    enum aes-cmac-96 {
      value 13;
      description
        "Generating 96-bit MAC using Advanced Encryption
         Standard (AES) Cipher-based Message Authentication
         Code (CMAC)";
      reference
        "RFC 4494:
           The AES-CMAC Algorithm and its Use with IPsec";
    }
    enum aes-cmac-128 {
      value 14;
      description
        "Generating 128-bit MAC using Advanced Encryption
         Standard (AES) Cipher-based Message Authentication
         Code (CMAC)";
      reference
        "RFC 4494:
           The AES-CMAC Algorithm and its Use with IPsec";
    }
```





```
    enum sha1-des3-kd {
      value 15;
      description
        "Generating MAC using triple DES encryption function";
```

```
          reference
            "RFC 3961:
               Encryption and Checksum Specifications for Kerberos
               5";
        }
      }
    }
    default "0";
    description
      "The uint16 filed shall be set by individual protocol
       families according to the mac algorithm value assigned by
       IANA. The setting is optional and by default is 0.  The
       enumeration filed is set to the selected mac algorithm.";
  }

  /*****************************************/
  /*   Identities for Encryption Algorithms   */
  /*****************************************/

  typedef encryption-algorithm-t {
    type union {
      type uint16;
      type enumeration {
        enum NONE {
          value 0;
          description
            "Encryption algorithm is NULL.";
        }
        enum aes-128-cbc {
          value 1;
          description
            "Encrypt message with AES algorithm in CBC mode with
             a key length of 128 bits.";
          reference
            "RFC 3565: Use of the Advanced Encryption Standard (AES)
             Encryption Algorithm in Cryptographic Message Syntax
             (CMS)";
        }
        enum aes-192-cbc {
          value 2;
          description
            "Encrypt message with AES algorithm in CBC mode with
             a key length of 192 bits";
          reference
```

```
        "RFC 3565: Use of the Advanced Encryption Standard (AES)
         Encryption Algorithm in Cryptographic Message Syntax
         (CMS)";
    }
    enum aes-256-cbc {
      value 3;
      description
        "Encrypt message with AES algorithm in CBC mode with
         a key length of 256 bits";
      reference
        "RFC 3565: Use of the Advanced Encryption Standard (AES)
         Encryption Algorithm in Cryptographic Message Syntax
         (CMS)";
    }
    enum aes-128-ctr {
      value 4;
      description
        "Encrypt message with AES algorithm in CTR mode with
         a key length of 128 bits";
      reference
        "RFC 3686:
           Using Advanced Encryption Standard (AES) Counter
           Mode with IPsec Encapsulating Security Payload
           (ESP)";
    }
    enum aes-192-ctr {
      value 5;
      description
        "Encrypt message with AES algorithm in CTR mode with
         a key length of 192 bits";
      reference
        "RFC 3686:
           Using Advanced Encryption Standard (AES) Counter
           Mode with IPsec Encapsulating Security Payload
           (ESP)";
    }
    enum aes-256-ctr {
      value 6;
      description
        "Encrypt message with AES algorithm in CTR mode with
         a key length of 256 bits";
      reference
        "RFC 3686:
           Using Advanced Encryption Standard (AES) Counter
           Mode with IPsec Encapsulating Security Payload
           (ESP)";
    }
```

```
          enum des3-cbc-sha1-kd {
```





```
            value 7;
            description
              "Encrypt message with 3DES algorithm in CBC mode
               with sha1 function for key derivation";
            reference
              "RFC 3961:
                 Encryption and Checksum Specifications for
                 Kerberos 5";
          }
          enum rc4-hmac {
            value 8;
            description
              "Encrypt message with rc4 algorithm";
            reference
              "RFC 4757:
                 The RC4-HMAC Kerberos Encryption Types Used by
                 Microsoft Windows";
          }
          enum rc4-hmac-exp {
            value 9;
            description
              "Encrypt message with rc4 algorithm that is exportable";
            reference
              "RFC 4757:
                 The RC4-HMAC Kerberos Encryption Types Used by
                 Microsoft Windows";
          }
        }
      }
      default "0";
      description
        "The uint16 filed shall be set by individual protocol
         families according to the encryption algorithm value
         assigned by IANA. The setting is optional and by default
         is 0.  The enumeration filed is set to the selected
         encryption algorithm.";
    }

    /***********************************************/
    /*   Identities for Encryption and MAC Algorithms   */
```

```
/*************************************************/

typedef encryption-and-mac-algorithm-t {
  type union {
    type uint16;
    type enumeration {
      enum NONE {
        value 0;
```





```
        description
          "Encryption and MAC algorithm is NULL.";
        reference
          "None";
      }
      enum aes-128-ccm {
        value 1;
        description
          "Encrypt message with AES algorithm in CCM
           mode with a key length of 128 bits; it can
           also be used for generating MAC";
        reference
          "RFC 4309: Using Advanced Encryption Standard
           (AES) CCM Mode with IPsec Encapsulating Security
           Payload (ESP)";
      }
      enum aes-192-ccm {
        value 2;
        description
          "Encrypt message with AES algorithm in CCM
           mode with a key length of 192 bits; it can
           also be used for generating MAC";
        reference
          "RFC 4309: Using Advanced Encryption Standard
           (AES) CCM Mode with IPsec Encapsulating Security
           Payload (ESP)";
      }
      enum aes-256-ccm {
        value 3;
        description
          "Encrypt message with AES algorithm in CCM
           mode with a key length of 256 bits; it can
           also be used for generating MAC";
```

```
      reference
        "RFC 4309: Using Advanced Encryption Standard
         (AES) CCM Mode with IPsec Encapsulating Security
         Payload (ESP)";
    }
    enum aes-128-gcm {
      value 4;
      description
        "Encrypt message with AES algorithm in GCM
         mode with a key length of 128 bits; it can
         also be used for generating MAC";
      reference
        "RFC 4106: The Use of Galois/Counter Mode (GCM)
         in IPsec Encapsulating Security Payload (ESP)";
    }
```





```
    enum aes-192-gcm {
      value 5;
      description
        "Encrypt message with AES algorithm in GCM
         mode with a key length of 192 bits; it can
         also be used for generating MAC";
      reference
        "RFC 4106: The Use of Galois/Counter Mode (GCM)
         in IPsec Encapsulating Security Payload (ESP)";
    }
    enum aes-256-gcm {
      value 6;
      description
        "Encrypt message with AES algorithm in GCM
         mode with a key length of 256 bits; it can
         also be used for generating MAC";
      reference
        "RFC 4106: The Use of Galois/Counter Mode (GCM)
         in IPsec Encapsulating Security Payload (ESP)";
    }
    enum chacha20-poly1305 {
      value 7;
      description
        "Encrypt message with chacha20 algorithm and generate
         MAC with POLY1305; it can also be used for generating
         MAC";
```

```
            reference
              "RFC 8439: ChaCha20 and Poly1305 for IETF Protocols";
          }
        }
      }
      default "0";
      description
        "The uint16 filed shall be set by individual protocol
         families according to the encryption and mac algorithm value
         assigned by IANA. The setting is optional and by default is
         0.  The enumeration filed is set to the selected encryption
         and mac algorithm.";
    }

    /*****************************************/
    /*   Identities for signature algorithm   */
    /*****************************************/

    typedef signature-algorithm-t {
      type union {
        type uint16;
        type enumeration {
```





```
          enum NONE {
            value 0;
            description
              "Signature algorithm is NULL";
          }
          enum dsa-sha1 {
            value 1;
            description
              "The signature algorithm using DSA algorithm with SHA1
               hash algorithm";
            reference
              "RFC 4253:
                 The Secure Shell (SSH) Transport Layer Protocol";
          }
          enum rsassa-pkcs1-sha1 {
            value 2;
            description
              "The signature algorithm using RSASSA-PKCS1-v1_5 with
               the SHA1 hash algorithm.";
```

```
        reference
          "RFC 4253:
             The Secure Shell (SSH) Transport Layer Protocol";
      }
      enum rsassa-pkcs1-sha256 {
        value 3;
        description
          "The signature algorithm using RSASSA-PKCS1-v1_5 with
           the SHA256 hash algorithm.";
        reference
          "RFC 8332:
             Use of RSA Keys with SHA-256 and SHA-512 in the
             Secure Shell (SSH) Protocol
           RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
      }
      enum rsassa-pkcs1-sha384 {
        value 4;
        description
          "The signature algorithm using RSASSA-PKCS1-v1_5 with
           the SHA384 hash algorithm.";
        reference
          "RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
      }
      enum rsassa-pkcs1-sha512 {
        value 5;
```





```
        description
          "The signature algorithm using RSASSA-PKCS1-v1_5 with
           the SHA512 hash algorithm.";
        reference
          "RFC 8332:
             Use of RSA Keys with SHA-256 and SHA-512 in the
             Secure Shell (SSH) Protocol
           RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
      }
      enum rsassa-pss-rsae-sha256 {
```

```
        value 6;
        description
          "The signature algorithm using RSASSA-PSS with mask
           generation function 1 and SHA256 hash algorithm. If
           the public key is carried in an X.509 certificate,
           it MUST use the rsaEncryption OID";
        reference
          "RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
      }
      enum rsassa-pss-rsae-sha384 {
        value 7;
        description
          "The signature algorithm using RSASSA-PSS with mask
           generation function 1 and SHA384 hash algorithm. If
           the public key is carried in an X.509 certificate,
           it MUST use the rsaEncryption OID";
        reference
          "RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
      }
      enum rsassa-pss-rsae-sha512 {
        value 8;
        description
          "The signature algorithm using RSASSA-PSS with mask
           generation function 1 and SHA512 hash algorithm. If
           the public key is carried in an X.509 certificate,
           it MUST use the rsaEncryption OID";
        reference
          "RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
      }
      enum rsassa-pss-pss-sha256 {
```





```
        value 9;
        description
          "The signature algorithm using RSASSA-PSS with mask
           generation function 1 and SHA256 hash algorithm. If
           the public key is carried in an X.509 certificate,
```

```
          it MUST use the rsaEncryption OID";
      reference
        "RFC 8446:
          The Transport Layer Security (TLS) Protocol
          Version 1.3";
    }
    enum rsassa-pss-pss-sha384 {
      value 10;
      description
        "The signature algorithm using RSASSA-PSS with mask
         generation function 1 and SHA384 hash algorithm. If
         the public key is carried in an X.509 certificate,
         it MUST use the rsaEncryption OID";
      reference
        "RFC 8446:
          The Transport Layer Security (TLS) Protocol
          Version 1.3";
    }
    enum rsassa-pss-pss-sha512 {
      value 11;
      description
        "The signature algorithm using RSASSA-PSS with mask
         generation function 1 and SHA512 hash algorithm. If
         the public key is carried in an X.509 certificate,
         it MUST use the rsaEncryption OID";
      reference
        "RFC 8446:
          The Transport Layer Security (TLS) Protocol
          Version 1.3";
    }
    enum ecdsa-secp256r1-sha256 {
      value 12;
      description
        "The signature algorithm using ECDSA with curve name
         secp256r1 and SHA256 hash algorithm.";
      reference
        "RFC 5656:
          Elliptic Curve Algorithm Integration in the Secure
          Shell Transport Layer
         RFC 8446:
          The Transport Layer Security (TLS) Protocol
          Version 1.3";
    }
```

```
enum ecdsa-secp384r1-sha384 {
  value 13;
  description
    "The signature algorithm using ECDSA with curve name
     secp384r1 and SHA384 hash algorithm.";
  reference
    "RFC 5656:
       Elliptic Curve Algorithm Integration in the Secure
       Shell Transport Layer
     RFC 8446:
       The Transport Layer Security (TLS) Protocol
       Version 1.3";
}
enum ecdsa-secp521r1-sha512 {
  value 14;
  description
    "The signature algorithm using ECDSA with curve name
     secp521r1 and SHA512 hash algorithm.";
  reference
    "RFC 5656:
       Elliptic Curve Algorithm Integration in the Secure
       Shell Transport Layer
     RFC 8446:
       The Transport Layer Security (TLS) Protocol
       Version 1.3";
}
enum ed25519 {
  value 15;
  description
    "The signature algorithm using EdDSA with curve x25519";
  reference
    "RFC 8032:
       Edwards-Curve Digital Signature Algorithm (EdDSA)";
}
enum ed25519-cts {
  value 16;
  description
    "The signature algorithm using EdDSA with curve x25519
     with phflag = 0";
  reference
    "RFC 8032:
       Edwards-Curve Digital Signature Algorithm (EdDSA)";
}
enum ed25519-ph {
  value 17;
  description
    "The signature algorithm using EdDSA with curve x25519
     with phflag = 1";
```

```
            reference
              "RFC 8032:
                 Edwards-Curve Digital Signature Algorithm (EdDSA)";
          }
          enum ed25519-sha512 {
            value 18;
            description
              "The signature algorithm using EdDSA with curve x25519
              and SHA-512 function";
            reference
              "RFC 8419:
                 Use of Edwards-Curve Digital Signature Algorithm
                 (EdDSA) Signatures in the Cryptographic Message
                 Syntax (CMS)";
          }
          enum ed448 {
            value 19;
            description
              "The signature algorithm using EdDSA with curve x448";
            reference
              "RFC 8032:
                 Edwards-Curve Digital Signature Algorithm (EdDSA)";
          }
          enum ed448-ph {
            value 20;
            description
              "The signature algorithm using EdDSA with curve x448
               and with PH being SHAKE256(x, 64) and phflag being 1";
            reference
              "RFC 8032:
                 Edwards-Curve Digital Signature Algorithm (EdDSA)";
          }
          enum ed448-shake256 {
            value 21;
            description
              "The signature algorithm using EdDSA with curve x448
              and SHAKE-256 function";
            reference
              "RFC 8419:
                 Use of Edwards-Curve Digital Signature Algorithm
                 (EdDSA) Signatures in the Cryptographic Message
                 Syntax (CMS)";
```

```
        }
        enum ed448-shake256-len {
          value 22;
          description
            "The signature algorithm using EdDSA with curve x448
            and SHAKE-256 function and a customized hash output";
```





```
          reference
            "RFC 8419:
               Use of Edwards-Curve Digital Signature Algorithm
               (EdDSA) Signatures in the Cryptographic Message
               Syntax (CMS)";
        }
        enum rsa-sha2-256 {
          value 23;
          description
            "The signature algorithm using RSA with SHA2 function
             for SSH protocol";
          reference
            "RFC 8332:
             Use of RSA Keys with SHA-256 and SHA-512
             in the Secure Shell (SSH) Protocol";
        }
        enum rsa-sha2-512 {
          value 24;
          description
            "The signature algorithm using RSA with SHA2 function
             for SSH protocol";
          reference
            "RFC 8332:
             Use of RSA Keys with SHA-256 and SHA-512
             in the Secure Shell (SSH) Protocol";
        }
        enum eccsi {
          value 25;
          description
            "The signature algorithm using ECCSI signature as
             defined in RFC 6507.";
          reference
            "RFC 6507:
               Elliptic Curve-Based Certificateless Signatures
               for Identity-based Encryption (ECCSI)";
```

```
        }
      }
    }
    default "0";
    description
      "The uint16 filed shall be set by individual protocol
       families according to the signature algorithm value
       assigned by IANA. The setting is optional and by default
       is 0.  The enumeration filed is set to the selected
       signature algorithm.";
  }

  /*********************************************/
```





```
  /*   Identities for key exchange algorithms   */
  /*********************************************/

  typedef key-exchange-algorithm-t {
    type union {
      type uint16;
      type enumeration {
        enum NONE {
          value 0;
          description
            "Key exchange algorithm is NULL.";
        }
        enum psk-only {
          value 1;
          description
            "Using Pre-shared key for authentication and key
             exchange";
          reference
            "RFC 4279:
               Pre-Shared Key cipher suites for Transport Layer
               Security (TLS)";
        }
        enum dhe-ffdhe2048 {
          value 2;
          description
            "Ephemeral Diffie Hellman key exchange with 2048 bit
             finite field";
          reference
```

```
        "RFC 7919:
           Negotiated Finite Field Diffie-Hellman Ephemeral
           Parameters for Transport Layer Security (TLS)";
      }
      enum dhe-ffdhe3072 {
        value 3;
        description
          "Ephemeral Diffie Hellman key exchange with 3072 bit
           finite field";
        reference
          "RFC 7919:
             Negotiated Finite Field Diffie-Hellman Ephemeral
             Parameters for Transport Layer Security (TLS)";
      }
      enum dhe-ffdhe4096 {
        value 4;
        description
          "Ephemeral Diffie Hellman key exchange with 4096 bit
           finite field";
        reference
```





```
        "RFC 7919:
           Negotiated Finite Field Diffie-Hellman Ephemeral
           Parameters for Transport Layer Security (TLS)";
      }
      enum dhe-ffdhe6144 {
        value 5;
        description
          "Ephemeral Diffie Hellman key exchange with 6144 bit
           finite field";
        reference
          "RFC 7919:
             Negotiated Finite Field Diffie-Hellman Ephemeral
             Parameters for Transport Layer Security (TLS)";
      }
      enum dhe-ffdhe8192 {
        value 6;
        description
          "Ephemeral Diffie Hellman key exchange with 8192 bit
           finite field";
        reference
          "RFC 7919:
```

```
            Negotiated Finite Field Diffie-Hellman Ephemeral
            Parameters for Transport Layer Security (TLS)";
        }
        enum psk-dhe-ffdhe2048 {
          value 7;
          description
            "Key exchange using pre-shared key with Diffie-Hellman
             key generation mechanism, where the DH group is
             FFDHE2048";
          reference
            "RFC 8446:
               The Transport Layer Security (TLS) Protocol
               Version 1.3";
        }
        enum psk-dhe-ffdhe3072 {
          value 8;
          description
            "Key exchange using pre-shared key with Diffie-Hellman
             key generation mechanism, where the DH group is
             FFDHE3072";
          reference
            "RFC 8446:
               The Transport Layer Security (TLS) Protocol
               Version 1.3";
        }
        enum psk-dhe-ffdhe4096 {
          value 9;
```





```
          description
            "Key exchange using pre-shared key with Diffie-Hellman
             key generation mechanism, where the DH group is
             FFDHE4096";
          reference
            "RFC 8446:
               The Transport Layer Security (TLS) Protocol
               Version 1.3";
        }
        enum psk-dhe-ffdhe6144 {
          value 10;
          description
            "Key exchange using pre-shared key with Diffie-Hellman
             key generation mechanism, where the DH group is
```

```
          FFDHE6144";
      reference
        "RFC 8446:
          The Transport Layer Security (TLS) Protocol
          Version 1.3";
    }
    enum psk-dhe-ffdhe8192 {
      value 11;
      description
        "Key exchange using pre-shared key with Diffie-Hellman
         key generation mechanism, where the DH group is
         FFDHE8192";
      reference
        "RFC 8446:
          The Transport Layer Security (TLS) Protocol
          Version 1.3";
    }
    enum ecdhe-secp256r1 {
      value 12;
      description
        "Ephemeral Diffie Hellman key exchange with elliptic
         group over curve secp256r1";
      reference
        "RFC 8422:
          Elliptic Curve Cryptography (ECC) Cipher Suites
          for Transport Layer Security (TLS) Versions 1.2
          and Earlier";
    }
    enum ecdhe-secp384r1 {
      value 13;
      description
        "Ephemeral Diffie Hellman key exchange with elliptic
         group over curve secp384r1";
      reference
```





```
        "RFC 8422:
          Elliptic Curve Cryptography (ECC) Cipher Suites
          for Transport Layer Security (TLS) Versions 1.2
          and Earlier";
    }
    enum ecdhe-secp521r1 {
      value 14;
```

```
        description
          "Ephemeral Diffie Hellman key exchange with elliptic
           group over curve secp521r1";
        reference
          "RFC 8422:
             Elliptic Curve Cryptography (ECC) Cipher Suites
             for Transport Layer Security (TLS) Versions 1.2
             and Earlier";
      }
      enum ecdhe-x25519 {
        value 15;
        description
          "Ephemeral Diffie Hellman key exchange with elliptic
           group over curve x25519";
        reference
          "RFC 8422:
             Elliptic Curve Cryptography (ECC) Cipher Suites
             for Transport Layer Security (TLS) Versions 1.2
             and Earlier";
      }
      enum ecdhe-x448 {
        value 16;
        description
          "Ephemeral Diffie Hellman key exchange with elliptic
           group over curve x448";
        reference
          "RFC 8422:
             Elliptic Curve Cryptography (ECC) Cipher Suites
             for Transport Layer Security (TLS) Versions 1.2
             and Earlier";
      }
      enum psk-ecdhe-secp256r1 {
        value 17;
        description
          "Key exchange using pre-shared key with elliptic
           group-based Ephemeral Diffie Hellman key exchange
           over curve secp256r1";
        reference
          "RFC 8446:
             The Transport Layer Security (TLS) Protocol
             Version 1.3";
```

```
}
enum psk-ecdhe-secp384r1 {
  value 18;
  description
    "Key exchange using pre-shared key with elliptic
     group-based Ephemeral Diffie Hellman key exchange
     over curve secp384r1";
  reference
    "RFC 8446:
       The Transport Layer Security (TLS) Protocol
       Version 1.3";
}
enum psk-ecdhe-secp521r1 {
  value 19;
  description
    "Key exchange using pre-shared key with elliptic
     group-based Ephemeral Diffie Hellman key exchange
     over curve secp521r1";
  reference
    "RFC 8446:
       The Transport Layer Security (TLS) Protocol
       Version 1.3";
}
enum psk-ecdhe-x25519 {
  value 20;
  description
    "Key exchange using pre-shared key with elliptic
     group-based Ephemeral Diffie Hellman key exchange
     over curve x25519";
  reference
    "RFC 8446:
       The Transport Layer Security (TLS) Protocol
       Version 1.3";
}
enum psk-ecdhe-x448 {
  value 21;
  description
    "Key exchange using pre-shared key with elliptic
     group-based Ephemeral Diffie Hellman key exchange
     over curve x448";
  reference
    "RFC 8446:
       The Transport Layer Security (TLS) Protocol
       Version 1.3";
}
enum diffie-hellman-group14-sha1 {
  value 22;
  description
```

```
        "Using DH group14 and SHA1 for key exchange";
      reference
        "RFC 4253:
           The Secure Shell (SSH) Transport Layer Protocol";
    }
    enum diffie-hellman-group14-sha256 {
      value 23;
      description
        "Using DH group14 and SHA-256 for key exchange";
      reference
        "RFC 8268:
          More Modular Exponentiation (MODP) Diffie-Hellman (DH)
          Key Exchange (KEX) Groups for Secure Shell (SSH)";
    }
    enum diffie-hellman-group15-sha512 {
      value 24;
      description
        "Using DH group15 and SHA-512 for key exchange";
      reference
        "RFC 8268:
          More Modular Exponentiation (MODP) Diffie-Hellman (DH)
          Key Exchange (KEX) Groups for Secure Shell (SSH)";
    }
    enum diffie-hellman-group16-sha512 {
      value 25;
      description
        "Using DH group16 and SHA-512 for key exchange";
      reference
        "RFC 8268:
          More Modular Exponentiation (MODP) Diffie-Hellman (DH)
          Key Exchange (KEX) Groups for Secure Shell (SSH)";
    }
    enum diffie-hellman-group17-sha512 {
      value 26;
      description
        "Using DH group17 and SHA-512 for key exchange";
      reference
        "RFC 8268:
          More Modular Exponentiation (MODP) Diffie-Hellman (DH)
          Key Exchange (KEX) Groups for Secure Shell (SSH)";
    }
    enum diffie-hellman-group18-sha512 {
      value 27;
```

```
        description
          "Using DH group18 and SHA-512 for key exchange";
        reference
          "RFC 8268:
            More Modular Exponentiation (MODP) Diffie-Hellman (DH)
```





```
            Key Exchange (KEX) Groups for Secure Shell (SSH)";
        }
        enum ecdh-sha2-secp256r1 {
          value 28;
          description
            "Elliptic curve-based Diffie Hellman key exchange over
             curve ecp256r1 and using SHA2 for MAC generation";
          reference
            "RFC 6239:
              Suite B Cryptographic Suites for Secure Shell (SSH)";
        }
        enum ecdh-sha2-secp384r1 {
          value 29;
          description
            "Elliptic curve-based Diffie Hellman key exchange over
             curve ecp384r1 and using SHA2 for MAC generation";
          reference
            "RFC 6239:
              Suite B Cryptographic Suites for Secure Shell (SSH)";
        }
        enum ecdh-x25519-x9.63-sha256 {
          value 30;
          description
            "Elliptic curve-based Diffie Hellman key exchange over
             curve x.25519 and using ANSI x9.63 with SHA256 as KDF";
          reference
            "RFC 8418:
              Use of the Elliptic Curve Diffie-Hellman Key Agreement
              Algorithm with X25519 and X448 in the Cryptographic
              Message Syntax (CMS)";
        }
        enum ecdh-x25519-x9.63-sha384 {
          value 31;
          description
            "Elliptic curve-based Diffie Hellman key exchange over
             curve x.25519 and using ANSI x9.63 with SHA384 as KDF";
```

```
        reference
          "RFC 8418:
             Use of the Elliptic Curve Diffie-Hellman Key Agreement
             Algorithm with X25519 and X448 in the Cryptographic
             Message Syntax (CMS)";
      }
      enum ecdh-x25519-x9.63-sha512 {
        value 32;
        description
          "Elliptic curve-based Diffie Hellman key exchange over
           curve x.25519 and using ANSI x9.63 with SHA512 as KDF";
        reference
```





```
          "RFC 8418:
             Use of the Elliptic Curve Diffie-Hellman Key Agreement
             Algorithm with X25519 and X448 in the Cryptographic
             Message Syntax (CMS)";
      }
      enum ecdh-x25519-hkdf-sha256 {
        value 33;
        description
          "Elliptic curve-based Diffie Hellman key exchange over
           curve x.25519 and using HKDF with SHA256 as KDF";
        reference
          "RFC 8418:
             Use of the Elliptic Curve Diffie-Hellman Key Agreement
             Algorithm with X25519 and X448 in the Cryptographic
             Message Syntax (CMS)";
      }
      enum ecdh-x25519-hkdf-sha384 {
        value 34;
        description
          "Elliptic curve-based Diffie Hellman key exchange over
           curve x.25519 and using HKDF with SHA384 as KDF";
        reference
          "RFC 8418:
             Use of the Elliptic Curve Diffie-Hellman Key Agreement
             Algorithm with X25519 and X448 in the Cryptographic
             Message Syntax (CMS)";
      }
      enum ecdh-x25519-hkdf-sha512 {
        value 35;
```

```
      description
        "Elliptic curve-based Diffie Hellman key exchange over
         curve x.25519 and using HKDF with SHA512 as KDF";
      reference
        "RFC 8418:
           Use of the Elliptic Curve Diffie-Hellman Key Agreement
           Algorithm with X25519 and X448 in the Cryptographic
           Message Syntax (CMS)";
    }
    enum ecdh-x448-x9.63-sha256 {
      value 36;
      description
        "Elliptic curve-based Diffie Hellman key exchange over
         curve x.448 and using ANSI x9.63 with SHA256 as KDF";
      reference
        "RFC 8418:
           Use of the Elliptic Curve Diffie-Hellman Key Agreement
           Algorithm with X25519 and X448 in the Cryptographic
           Message Syntax (CMS)";
```





```
    }
    enum ecdh-x448-x9.63-sha384 {
      value 37;
      description
        "Elliptic curve-based Diffie Hellman key exchange over
         curve x.448 and using ANSI x9.63 with SHA384 as KDF";
      reference
        "RFC 8418:
           Use of the Elliptic Curve Diffie-Hellman Key Agreement
           Algorithm with X25519 and X448 in the Cryptographic
           Message Syntax (CMS)";
    }
    enum ecdh-x448-x9.63-sha512 {
      value 38;
      description
        "Elliptic curve-based Diffie Hellman key exchange over
         curve x.448 and using ANSI x9.63 with SHA512 as KDF";
      reference
        "RFC 8418:
           Use of the Elliptic Curve Diffie-Hellman Key Agreement
           Algorithm with X25519 and X448 in the Cryptographic
           Message Syntax (CMS)";
```

```
        }
        enum ecdh-x448-hkdf-sha256 {
          value 39;
          description
            "Elliptic curve-based Diffie Hellman key exchange over
             curve x.448 and using HKDF with SHA256 as KDF";
          reference
            "RFC 8418:
               Use of the Elliptic Curve Diffie-Hellman Key Agreement
               Algorithm with X25519 and X448 in the Cryptographic
               Message Syntax (CMS)";
        }
        enum ecdh-x448-hkdf-sha384 {
          value 40;
          description
            "Elliptic curve-based Diffie Hellman key exchange over
             curve x.448 and using HKDF with SHA384 as KDF";
          reference
            "RFC 8418:
               Use of the Elliptic Curve Diffie-Hellman Key Agreement
               Algorithm with X25519 and X448 in the Cryptographic
               Message Syntax (CMS)";
        }
        enum ecdh-x448-hkdf-sha512 {
          value 41;
          description
```





```
            "Elliptic curve-based Diffie Hellman key exchange over
             curve x.448 and using HKDF with SHA512 as KDF";
          reference
            "RFC 8418:
               Use of the Elliptic Curve Diffie-Hellman Key Agreement
               Algorithm with X25519 and X448 in the Cryptographic
               Message Syntax (CMS)";
        }

        enum rsaes-oaep {
          value 42;
          description
            "RSAES-OAEP combines the RSAEP and RSADP primitives with
             the EME-OAEP encoding method";
          reference
```

```
            "RFC 8017:
               PKCS #1:
                 RSA Cryptography Specifications Version 2.2.";
          }
          enum rsaes-pkcs1-v1_5 {
            value 43;
            description
              "RSAES-PKCS1-v1_5 combines the RSAEP and RSADP
               primitives with the EME-PKCS1-v1_5 encoding method";
            reference
              "RFC 8017:
                 PKCS #1:
                   RSA Cryptography Specifications Version 2.2.";
          }
        }
      }
      default "0";
      description
        "The uint16 filed shall be set by individual protocol
         families according to the key exchange algorithm value
         assigned by IANA. The setting is optional and by default
         is 0.  The enumeration filed is set to the selected key
         exchange algorithm.";
    }

    /*********************************************/
    /*   Identities for Key Format Structures    */
    /*********************************************/

    /*** all key format types ****/

    identity key-format-base {
      description "Base key-format identity for all keys.";
```





```
    }

    identity public-key-format {
      base "key-format-base";
      description "Base key-format identity for public keys.";
    }

    identity private-key-format {
```

```
    base "key-format-base";
    description "Base key-format identity for private keys.";
  }

  identity symmetric-key-format {
    base "key-format-base";
    description "Base key-format identity for symmetric keys.";
  }


  /**** for private keys ****/

  identity rsa-private-key-format {
      base "private-key-format";
      description "An RSAPrivateKey (from RFC 3447).";
  }

  identity ec-private-key-format {
      base "private-key-format";
      description "An ECPrivateKey (from RFC 5915)";
  }

  identity one-asymmetric-key-format {
      base "private-key-format";
      description "A OneAsymmetricKey (from RFC 5958).";
  }

  identity encrypted-private-key-format {
      base "private-key-format";
      description
        "A CMS EncryptedData structure (RFC 5652)
         containing a OneAsymmetricKey (RFC 5958).";
   }


  /**** for public keys ****/

  identity ssh-public-key-format {
      base "public-key-format";
      description
```





```
        "The public key format described by RFC 4716.";
```

```
}

identity subject-public-key-info-format {
    base "public-key-format";
    description
      "A SubjectPublicKeyInfo (from RFC 5280).";
}


/**** for symmetric keys ****/

identity octet-string-key-format {
    base "symmetric-key-format";
    description "An OctetString from ASN.1.";
      /*
      // Knowing that it is an "OctetString" isn't really helpful.
      // Knowing the length of the octet string would be helpful,
      // as it relates to the algorithm's block size.  We may want
      // to only (for now) use "one-symmetric-key-format" for
      // symmetric keys...were the usability issues Juergen
      // mentioned before only apply to asymmetric keys?
      */
}

identity one-symmetric-key-format {
    base "symmetric-key-format";
    description "A OneSymmetricKey (from RFC6031).";
}

identity encrypted-symmetric-key-format {
    base "symmetric-key-format";
    description
      "A CMS EncryptedData structure (RFC 5652)
       containing a OneSymmetricKey (RFC 6031).";
}



/*************************************************/
/*   Typedefs for ASN.1 structures from RFC 5280   */
/*************************************************/

typedef x509 {
  type binary;
  description
    "A Certificate structure, as specified in RFC 5280,
     encoded using 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).";
    }

    typedef crl {
      type binary;
      description
        "A CertificateList structure, as specified in RFC 5280,
         encoded using 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).";
    }

    /*********************************************/
    /*   Typedefs for ASN.1 structures from 5652   */
    /*********************************************/

    typedef cms {
      type binary;
      description
        "A ContentInfo structure, as specified in RFC 5652,
         encoded using ASN.1 distinguished encoding rules (DER),
         as specified in ITU-T X.690.";
      reference
        "RFC 5652:
           Cryptographic Message Syntax (CMS)
         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).";
      }
```





```
      typedef data-content-cms {
        type cms;
        description
          "A CMS structure whose top-most content type MUST be the
           data content type, as described by Section 4 in RFC 5652.";
        reference
          "RFC 5652: Cryptographic Message Syntax (CMS)";
      }

      typedef signed-data-cms {
        type cms;
        description
          "A CMS structure whose top-most content type MUST be the
           signed-data content type, as described by Section 5 in
           RFC 5652.";
        reference
          "RFC 5652: Cryptographic Message Syntax (CMS)";
      }

      typedef enveloped-data-cms {
        type cms;
        description
          "A CMS structure whose top-most content type MUST be the
           enveloped-data content type, as described by Section 6
           in RFC 5652.";
        reference
          "RFC 5652: Cryptographic Message Syntax (CMS)";
      }

      typedef digested-data-cms {
        type cms;
        description
          "A CMS structure whose top-most content type MUST be the
           digested-data content type, as described by Section 7
           in RFC 5652.";
        reference
          "RFC 5652: Cryptographic Message Syntax (CMS)";
      }
```

```
      typedef encrypted-data-cms {
        type cms;
        description
          "A CMS structure whose top-most content type MUST be the
           encrypted-data content type, as described by Section 8
           in RFC 5652.";
        reference
          "RFC 5652: Cryptographic Message Syntax (CMS)";
      }
```





```
      typedef authenticated-data-cms {
        type cms;
        description
          "A CMS structure whose top-most content type MUST be the
           authenticated-data content type, as described by Section 9
           in RFC 5652.";
        reference
          "RFC 5652: Cryptographic Message Syntax (CMS)";
      }

      /****************************************************/
      /*   Typedefs for structures related to RFC 4253    */
      /****************************************************/

      typedef ssh-host-key {
        type binary;
        description
          "The binary public key data for this SSH key, as
           specified by RFC 4253, Section 6.6, i.e.:

             string    certificate or public key format
                       identifier
             byte[n]   key/certificate data.";
        reference
          "RFC 4253: The Secure Shell (SSH) Transport Layer
                     Protocol";
      }

      /*********************************************************/
      /*   Typedefs for ASN.1 structures related to RFC 5280   */
      /*********************************************************/
```

```
typedef trust-anchor-cert-x509 {
  type x509;
  description
    "A Certificate structure that MUST encode a self-signed
     root certificate.";
}

typedef end-entity-cert-x509 {
  type x509;
  description
    "A Certificate structure that MUST encode a certificate
     that is neither self-signed nor having Basic constraint
     CA true.";
}

/****************************************************/
```





```
/*   Typedefs for ASN.1 structures related to RFC 5652   */
/****************************************************/

typedef trust-anchor-cert-cms {
  type signed-data-cms;
  description
    "A CMS SignedData structure that MUST contain the chain of
     X.509 certificates needed to authenticate the certificate
     presented by a client or end-entity.

     The CMS MUST contain only a single chain of certificates.
     The client or end-entity certificate MUST only authenticate
     to last intermediate CA certificate listed in the chain.

     In all cases, the chain MUST include a self-signed root
     certificate.  In the case where the root certificate is
     itself the issuer of the client or end-entity certificate,
     only one certificate is present.

     This CMS structure MAY (as applicable where this type is
     used) also contain suitably fresh (as defined by local
     policy) revocation objects with which the device can
     verify the revocation status of the certificates.

     This CMS encodes the degenerate form of the SignedData
```

```
        structure that is commonly used to disseminate X.509
        certificates and revocation objects (RFC 5280).";
    reference
      "RFC 5280:
        Internet X.509 Public Key Infrastructure Certificate
        and Certificate Revocation List (CRL) Profile.";
  }

  typedef end-entity-cert-cms {
    type signed-data-cms;
    description
      "A CMS SignedData structure that MUST contain the end
       entity certificate itself, and MAY contain any number
       of intermediate certificates leading up to a trust
       anchor certificate.  The trust anchor certificate
       MAY be included as well.

       The CMS MUST contain a single end entity certificate.
       The CMS MUST NOT contain any spurious certificates.

       This CMS structure MAY (as applicable where this type is
       used) also contain suitably fresh (as defined by local
       policy) revocation objects with which the device can
```





```
       verify the revocation status of the certificates.

       This CMS encodes the degenerate form of the SignedData
       structure that is commonly used to disseminate X.509
       certificates and revocation objects (RFC 5280).";
    reference
      "RFC 5280:
        Internet X.509 Public Key Infrastructure Certificate
        and Certificate Revocation List (CRL) Profile.";
  }

  typedef ssh-public-key-type {  // DELETE?
     type binary;
     description
       "The binary public key data for this SSH key, as
        specified by RFC 4253, Section 6.6, i.e.:

          string    certificate or public key format
```

```
                    identifier
          byte[n]   key/certificate data.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport
                   Layer Protocol";
    }

    /*********************************************/
    /*   Groupings for keys and/or certificates   */
    /*********************************************/

    grouping symmetric-key-grouping {
      description
        "A symmetric key and algorithm.";
      leaf algorithm {
        type encryption-algorithm-t;
        mandatory true;
        description
          "The algorithm to be used when generating the key.";
        reference
          "RFC CCCC: Common YANG Data Types for Cryptography";
      }
      leaf key-format {
        nacm:default-deny-write;
        when "../key";
        type identityref {
          base symmetric-key-format;
        }
        description "Identifies the symmetric key's format.";
      }
```





```
      choice key-type {
        mandatory true;
        description
          "Choice between key types.";
        leaf key {
          nacm:default-deny-all;
          type binary;
          //must "../key-format";  FIXME: remove comment if approach ok
          description
            "The binary value of the key.  The interpretation of
             the value is defined by 'key-format'.  For example,
```

```
            FIXME.";
          reference
            "RFC XXXX: FIXME";
        }
        leaf hidden-key {
          nacm:default-deny-write;
          type empty;
          description
            "A permanently hidden key.  How such keys are created
             is outside the scope of this module.";
        }
      }
    }

    grouping public-key-grouping {
      description
        "A public key and its associated algorithm.";
      leaf algorithm {
        nacm:default-deny-write;
        type asymmetric-key-algorithm-t;
        mandatory true;
        description
          "Identifies the key's algorithm.";
        reference
          "RFC CCCC: Common YANG Data Types for Cryptography";
      }
      leaf public-key-format {
        nacm:default-deny-write;
        when "../public-key";
        type identityref {
          base public-key-format;
        }
        description "Identifies the key's format.";
      }
      leaf public-key {
        nacm:default-deny-write;
        type binary;
```





```
        //must "../public-key-format"; FIXME: rm comment if approach ok
        mandatory true;
        description
          "The binary value of the public key.  The interpretation
```

```
        of the value is defined by 'public-key-format' field.";
    }
  }

  grouping asymmetric-key-pair-grouping {
    description
      "A private key and its associated public key and algorithm.";
    uses public-key-grouping;
    leaf private-key-format {
      nacm:default-deny-write;
      when "../private-key";
      type identityref {
        base private-key-format;
      }
      description "Identifies the key's format.";
    }
    choice private-key-type {
      mandatory true;
      description
        "Choice between key types.";
      leaf private-key {
        nacm:default-deny-all;
        type binary;
        //must "../private-key-format"; FIXME: rm comment if ok
        description
          "The value of the binary key.  The key's value is
           interpreted by the 'private-key-format' field.";
      }
      leaf hidden-private-key {
        nacm:default-deny-write;
        type empty;
        description
          "A permanently hidden key.  How such keys are created
           is outside the scope of this module.";
      }
    }
  }

  grouping trust-anchor-cert-grouping {
    description
      "A trust anchor certificate, and a notification for when
       it is about to (or already has) expire.";
    leaf cert {
      nacm:default-deny-write;
```

```
        type trust-anchor-cert-cms;
        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 trust-anchor-certs-grouping {
      description
        "A list of trust anchor certificates, and a notification
         for when one is about to (or already has) expire.";
      leaf-list cert {
        nacm:default-deny-write;
        type trust-anchor-cert-cms;
        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 end-entity-cert-grouping {
      description
        "An end entity certificate, and a notification for when
         it is about to (or already has) expire.  Implementations
         SHOULD assert that, where used, the end entity certificate
         contains the expected public key.";
      leaf cert {
        nacm:default-deny-write;
        type end-entity-cert-cms;
        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 end-entity-certs-grouping {
      description
        "A list of end entity certificates, and a notification for
         when one is about to (or already has) expire.";
      leaf-list cert {
```

```
        nacm:default-deny-write;
        type end-entity-cert-cms;
        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-cert-grouping {
      description
        "A private/public key pair and an associated certificate.
         Implementations SHOULD assert that certificates contain
         the matching public key.";
      uses asymmetric-key-pair-grouping;
      uses end-entity-cert-grouping;
      action generate-certificate-signing-request {
        nacm:default-deny-all;
        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 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; // FIXME: does this need to be mandatory?
            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).";
          }
        }
      } // generate-certificate-signing-request
    } // asymmetric-key-pair-with-cert-grouping

    grouping asymmetric-key-pair-with-certs-grouping {
      description
        "A private/public key pair and associated certificates.
```





```
         Implementations SHOULD assert that certificates contain
         the matching public key.";
      uses asymmetric-key-pair-grouping;
      container certificates {
        nacm:default-deny-write;
        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 {
          key "name";
          description
            "A certificate for this asymmetric key.";
          leaf name {
            type string;
            description
              "An arbitrary name for the certificate.  If the name
               matches the name of a certificate that exists
```

```
            independently in <operational> (i.e., an IDevID),
            then the 'cert' node MUST NOT be configured.";
        }
        uses end-entity-cert-grouping;
      }
    } // certificates
    action generate-certificate-signing-request {
      nacm:default-deny-all;
      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 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; // FIXME: does this need to be mandatory?
          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).";
            }
          }
        } // generate-certificate-signing-request
      } // asymmetric-key-pair-with-certs-grouping
    }
```





<CODE ENDS>

## 3. Security Considerations

### 3.1. Support for Algorithms

In order to use YANG identities for algorithm identifiers, only the most commonly used RSA key lengths are supported for the RSA algorithm. Additional key lengths can be defined in another module or added into a future version of this document.

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

## 3.2. No Support for CRMF

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. If it is desired to support CRMF in the future, a backwards compatible solution can be defined at that time.

## 3.3. Access to Data Nodes

The YANG module in this document defines "grouping" statements that are 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.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

There are a number of data nodes defined by the grouping statements that are writable/creatable/deletable (i.e., config true, which is the default). Some of 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:

*: All of the data nodes defined by all the groupings are considered sensitive to write operations. For instance, the modification of a public key or a certificate can dramatically alter the implemented security policy. For this reason, the NACM extension "default-deny-write" has been applied to all the data nodes defined by all the groupings.

Some of the readable data nodes in the 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:

/private-key: The "private-key" node defined in the "asymmetric-key-pair-grouping" grouping is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. For this reason, the NACM extension "default-deny-all" has been applied to it here.

Some of the 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:

*: All of the "action" statements defined by groupings SHOULD only be executed by authorized users. For this reason, the NACM extension "default-deny-all" has been applied to all of them. Note that NACM uses "default-deny-all" to protect "RPC" and "action" statements; it does not define, e.g., an extension called "default-deny-execute".

generate-certificate-signing-request: For this action, 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 when the secure transport layer was established.

# 4. IANA Considerations

## 4.1. The IETF XML Registry

This document registers one URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

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

### 4.2. 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-crypto-types
namespace:    urn:ietf:params:xml:ns:yang:ietf-crypto-types
prefix:       ct
reference:    RFC XXXX
```

## 5. References

### 5.1. Normative References

[ITU.X690.2015]
International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/>.

[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>.

[RFC2404] Madson, C. and R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, DOI 10.17487/RFC2404, November 1998, <https://www.rfc-editor.org/info/rfc2404>.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, DOI 10.17487/RFC3565, July 2003, <https://www.rfc-editor.org/info/rfc3565>.

[RFC3686] Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", RFC 3686, DOI 10.17487/RFC3686, January 2004, <https://www.rfc-editor.org/info/rfc3686>.

[RFC4106] Viega, J. and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)", RFC 4106, DOI 10.17487/RFC4106, June 2005, <https://www.rfc-editor.org/info/rfc4106>.





[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <https://www.rfc-editor.org/info/rfc4279>.

[RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, DOI 10.17487/RFC4309, December 2005, <https://www.rfc-editor.org/info/rfc4309>.

[RFC4494] Song, JH., Poovendran, R., and J. Lee, "The AES-CMAC-96 Algorithm and Its Use with IPsec", RFC 4494, DOI 10.17487/RFC4494, June 2006, <https://www.rfc-editor.org/info/rfc4494>.

[RFC4543] McGrew, D. and J. Viega, "The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH", RFC 4543, DOI 10.17487/RFC4543, May 2006, <https://www.rfc-editor.org/info/rfc4543>.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <https://www.rfc-editor.org/info/rfc4868>.

[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, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, DOI 10.17487/RFC5656, December 2009, <https://www.rfc-editor.org/info/rfc5656>.

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, DOI 10.17487/RFC6187, March 2011, <https://www.rfc-editor.org/info/rfc6187>.





[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7919] Gillmor, D., "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)", RFC 7919, DOI 10.17487/RFC7919, August 2016, <https://www.rfc-editor.org/info/rfc7919>.

[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>.

[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>.

[RFC8268] Baushke, M., "More Modular Exponentiation (MODP) Diffie-Hellman (DH) Key Exchange (KEX) Groups for Secure Shell (SSH)", RFC 8268, DOI 10.17487/RFC8268, December 2017, <https://www.rfc-editor.org/info/rfc8268>.

[RFC8332] Bider, D., "Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol", RFC 8332, DOI 10.17487/RFC8332, March 2018, <https://www.rfc-editor.org/info/rfc8332>.

[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>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

## 5.2. Informative References

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





[RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <https://www.rfc-editor.org/info/rfc3174>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc4211>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

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

[RFC5915] Turner, S. and D. Brown, "Elliptic Curve Private Key Structure", RFC 5915, DOI 10.17487/RFC5915, June 2010, <https://www.rfc-editor.org/info/rfc5915>.

[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>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6239] Igoe, K., "Suite B Cryptographic Suites for Secure Shell (SSH)", RFC 6239, DOI 10.17487/RFC6239, May 2011, <https://www.rfc-editor.org/info/rfc6239>.





[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>.

[RFC6507] Groves, M., "Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI)", RFC 6507, DOI 10.17487/RFC6507, February 2012, <https://www.rfc-editor.org/info/rfc6507>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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>.

[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>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.





# Appendix A. Examples

## A.1. The "asymmetric-key-pair-with-certs-grouping" Grouping

The following example module illustrates the use of both the "symmetric-key-grouping" and the "asymmetric-key-pair-with-certs-grouping" groupings defined in the "ietf-crypto-types" module.

```
module ex-crypto-types-usage {
  yang-version 1.1;

  namespace "http://example.com/ns/example-crypto-types-usage";
  prefix "ectu";

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

  organization
   "Example Corporation";

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

  description
   "This module illustrates the grouping
    defined in the crypto-types draft called
    'asymmetric-key-pair-with-certs-grouping'.";

  revision "1001-01-01" {
    description
     "Initial version";
    reference
     "RFC ????: Usage Example for RFC XXXX";
  }

  container symmetric-keys {
    description
      "A container of symmetric keys.";
    list symmetric-key {
      key name;
      description
        "A symmetric key";
      leaf name {
        type string;
        description
```

```
          "An arbitrary name for this key.";
      }
      uses ct:symmetric-key-grouping;
    }
  }
  container asymmetric-keys {
    description
      "A container of asymmetric keys.";
    list asymmetric-key {
      key name;
      leaf name {
        type string;
        description
          "An arbitrary name for this key.";
      }
      uses ct:asymmetric-key-pair-with-certs-grouping;
      description
        "An asymmetric key pair with associated certificates.";
    }
  }
}
```

Given the above example usage module, the following example illustrates some configured keys.

```
========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

<symmetric-keys xmlns="http://example.com/ns/example-crypto-types-us\
age"
      xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
  <symmetric-key>
    <name>ex-symmetric-key</name>
    <algorithm>aes-256-cbc</algorithm>
    <key-format>ct:octet-string-key-format</key-format>
    <key>base64encodedvalue==</key>
  </symmetric-key>
  <symmetric-key>
    <name>ex-hidden-symmetric-key</name>
    <algorithm>aes-256-cbc</algorithm>
    <hidden-key/>
  </symmetric-key>
</symmetric-keys>

<asymmetric-keys xmlns="http://example.com/ns/example-crypto-types-u\
sage"
      xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
  <asymmetric-key>
    <name>ex-asymmetric-key</name>
```

```
      <algorithm>rsa2048</algorithm>
      <public-key-format>ct:subject-public-key-info-format</public-key\
  -format>
      <public-key>base64encodedvalue==</public-key>
      <private-key-format>ct:rsa-private-key-format</private-key-forma\
  t>
      <private-key>base64encodedvalue==</private-key>
      <certificates>
        <certificate>
          <name>ex-cert</name>
          <cert>base64encodedvalue==</cert>
        </certificate>
      </certificates>
    </asymmetric-key>
    <asymmetric-key>
      <name>ex-hidden-asymmetric-key</name>
      <algorithm>rsa2048</algorithm>
      <public-key-format>ct:subject-public-key-info-format</public-key\
  -format>
      <public-key>base64encodedvalue==</public-key>
      <hidden-private-key/>
      <certificates>
        <certificate>
          <name>ex-hidden-key-cert</name>
          <cert>base64encodedvalue==</cert>
        </certificate>
      </certificates>
    </asymmetric-key>
  </asymmetric-keys>
```

## A.2. The "generate-certificate-signing-request" Action

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

```
   REQUEST

   ========== NOTE: '\' line wrapping per BCP XXX (RFC XXXX) ===========

   <rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <action xmlns="urn:ietf:params:xml:ns:yang:1">
       <asymmetric-keys xmlns="http://example.com/ns/example-crypto-typ\
   es-usage">
         <asymmetric-key>
           <name>ex-key-sect571r1</name>
           <generate-certificate-signing-request>
             <subject>base64encodedvalue==</subject>
             <attributes>base64encodedvalue==</attributes>
           </generate-certificate-signing-request>
         </asymmetric-key>
       </asymmetric-keys>
     </action>
   </rpc>

   RESPONSE

   <rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <certificate-signing-request
        xmlns="http://example.com/ns/example-crypto-types-usage">
        base64encodedvalue==
      </certificate-signing-request>
   </rpc-reply>
```

## A.3. The "certificate-expiration" Notification

The following example illustrates the "certificate-expiration" notification in use with the NETCONF protocol.

```
   <notification
     xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
     <eventTime>2018-05-25T00:01:00Z</eventTime>
     <keys xmlns="http://example.com/ns/example-crypto-types-usage">
       <key>
         <name>locally-defined key</name>
         <certificates>
           <certificate>
             <name>my-cert</name>
             <certificate-expiration>
               <expiration-date>
                 2018-08-05T14:18:53-05:00
               </expiration-date>
             </certificate-expiration>
           </certificate>
         </certificates>
       </key>
     </keys>
   </notification>
```

## Appendix B. Change Log

### B.1. I-D to 00

- Removed groupings and notifications.
- Added typedefs for identityrefs.
- Added typedefs for other RFC 5280 structures.

- Added typedefs for other RFC 5652 structures.

- Added convenience typedefs for RFC 4253, RFC 5280, and RFC 5652.

### B.2. 00 to 01

- Moved groupings from the draft-ietf-netconf-keystore here.

### B.3. 01 to 02

- Removed unwanted "mandatory" and "must" statements.

- Added many new crypto algorithms (thanks Haiguang!)

- Clarified in asymmetric-key-pair-with-certs-grouping, in certificates/certificate/name/description, that if the name MUST NOT match the name of a certificate that exists independently in





<operational>, enabling certs installed by the manufacturer (e.g., an IDevID).

### B.4. 02 to 03

- renamed base identity 'asymmetric-key-encryption-algorithm' to 'asymmetric-key-algorithm'.

- added new 'asymmetric-key-algorithm' identities for secp192r1, secp224r1, secp256r1, secp384r1, and secp521r1.

- removed 'mac-algorithm' identities for mac-aes-128-ccm, mac-aes-192-ccm, mac-aes-256-ccm, mac-aes-128-gcm, mac-aes-192-gcm, mac-aes-256-gcm, and mac-chacha20-poly1305.

- for all -cbc and -ctr identities, renamed base identity 'symmetric-key-encryption-algorithm' to 'encryption-algorithm'.

- for all -ccm and -gcm identities, renamed base identity 'symmetric-key-encryption-algorithm' to 'encryption-and-mac-algorithm' and renamed the identity to remove the "enc-" prefix.

- for all the 'signature-algorithm' based identities, renamed from 'rsa-*' to 'rsassa-*'.
- removed all of the "x509v3-" prefixed 'signature-algorithm' based identities.
- added 'key-exchange-algorithm' based identities for 'rsaes-oaep' and 'rsaes-pkcs1-v1_5'.
- renamed typedef 'symmetric-key-encryption-algorithm-ref' to 'symmetric-key-algorithm-ref'.
- renamed typedef 'asymmetric-key-encryption-algorithm-ref' to 'asymmetric-key-algorithm-ref'.
- added typedef 'encryption-and-mac-algorithm-ref'.
- Updated copyright date, boilerplate template, affiliation, and folding algorithm.

### B.5. 03 to 04

- ran YANG module through formatter.





### B.6. 04 to 05

- fixed broken symlink causing reformatted YANG module to not show.

### B.7. 05 to 06

- Added NACM annotations.
- Updated Security Considerations section.
- Added 'asymmetric-key-pair-with-cert-grouping' grouping.
- Removed text from 'permanently-hidden' enum regarding such keys not being backed up or restored.

- Updated the boilerplate text in module-level "description" statement to match copyeditor convention.

- Added an explanation to the 'public-key-grouping' and 'asymmetric-key-pair-grouping' statements as for why the nodes are not mandatory (e.g., because they may exist only in <operational>.

- Added 'must' expressions to the 'public-key-grouping' and 'asymmetric-key-pair-grouping' statements ensuring sibling nodes are either all exist or do not all exist.

- Added an explanation to the 'permanently-hidden' that the value cannot be configured directly by clients and servers MUST fail any attempt to do so.

- Added 'trust-anchor-certs-grouping' and 'end-entity-certs-grouping' (the plural form of existing groupings).

- Now states that keys created in <operational> by the *-hidden-key actions are bound to the lifetime of the parent 'config true' node, and that subsequent invocations of either action results in a failure.

### B.8. 06 to 07

- Added clarifications that implementations SHOULD assert that configured certificates contain the matching public key.

- Replaced the 'generate-hidden-key' and 'install-hidden-key' actions with special 'crypt-hash' -like input/output values.





### B.9. 07 to 08

- Removed the 'generate-key and 'hidden-key' features.

- Added grouping symmetric-key-grouping

- Modified 'asymmetric-key-pair-grouping' to have a 'choice' statement for the keystone module to augment into, as well as

replacing the 'union' with leafs (having different NACM settings.

### B.10. 08 to 09

- Converting algorithm from identities to enumerations.

### B.11. 09 to 10

- All of the below changes are to the algorithm enumerations defined in ietf-crypto-types.
- Add in support for key exchange over x.25519 and x.448 based on RFC 8418.
- Add in SHAKE-128, SHAKE-224, SHAKE-256, SHAKE-384 and SHAKE 512
- Revise/add in enum of signature algorithm for x25519 and x448
- Add in des3-cbc-sha1 for IPSec
- Add in sha1-des3-kd for IPSec
- Add in definit for rc4-hmac and rc4-hmac-exp. These two algorithms have been deprecated in RFC 8429. But some existing draft in i2nsf may still want to use them.
- Add x25519 and x448 curve for asymmetric algorithms
- Add signature algorithms ed25519, ed25519-cts, ed25519ph
- add signature algorithms ed448, ed448ph
- Add in rsa-sha2-256 and rsa-sha2-512 for SSH protocols (rfc8332)

### B.12. 10 to 11

- Added a "key-format" identity.
- Added symmetric keys to the example in Appendix A.





Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Martin Bjorklund, Nick Hancock, Balazs Kovacs, Juergen Schoenwaelder, Eric Voit, and Liang Xia.

## Authors' Addresses

Kent Watsen
Watsen Networks

EMail: kent+ietf@watsen.net

Wang Haiguang
Huawei

EMail: wang.haiguang.shieldlab@huawei.com