Stream: Internet Engineering Task Force (IETF)
RFC: 9644
Category: Standards Track
Published: September 2024
ISSN: 2070-1721
Author: K. Watsen
Watsen Networks

RFC 9644
YANG Groupings for SSH Clients and SSH Servers

Abstract

This document presents three IETF-defined YANG modules and a script used to create four supporting IANA modules.

The three IETF modules are ietf-ssh-common, ietf-ssh-client, and ietf-ssh-server. The "ietf-ssh-client" and "ietf-ssh-server" modules are the primary productions of this work, supporting the configuration and monitoring of Secure Shell (SSH) clients and servers.

The four IANA modules are iana-ssh-encryption-algs, iana-ssh-key-exchange-algs, iana-ssh-mac-algs, and iana-ssh-public-key-algs. These modules each define YANG enumerations providing support for an IANA-maintained algorithm registry.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9644.

Copyright Notice

Table of Contents

1. Introduction 3
   1.1. Regarding the Three IETF Modules 4
   1.2. Relation to Other RFCs 4
   1.3. Specification Language 5
   1.4. Adherence to the NMDA 6
   1.5. Conventions 6
2. The "ietf-ssh-common" Module 6
   2.1. Data Model Overview 6
   2.2. Example Usage 8
   2.3. YANG Module 11
3. The "ietf-ssh-client" Module 20
   3.1. Data Model Overview 20
   3.2. Example Usage 22
   3.3. YANG Module 24
4. The "ietf-ssh-server" Module 31
   4.1. Data Model Overview 31
   4.2. Example Usage 33
   4.3. YANG Module 36
5. Security Considerations 44
   5.1. Considerations for the "iana-ssh-key-exchange-algs" Module 44
   5.2. Considerations for the "iana-ssh-encryption-algs" Module 45
   5.3. Considerations for the "iana-ssh-mac-algs" Module 45
   5.4. Considerations for the "iana-ssh-public-key-algs" Module 45
   5.5. Considerations for the "ietf-ssh-common" YANG Module 46
   5.6. Considerations for the "ietf-ssh-client" YANG Module 47
   5.7. Considerations for the "ietf-ssh-server" YANG Module 47

6. IANA Considerations 48
6.1. The IETF XML Registry 48
6.2. The YANG Module Names Registry 49
6.3. Considerations for the "iana-ssh-encryption-algs" Module 50
6.4. Considerations for the "iana-ssh-mac-algs" Module 51
6.5. Considerations for the "iana-ssh-public-key-algs" Module 52
6.6. Considerations for the "iana-ssh-key-exchange-algs" Module 54
7. References 55
7.1. Normative References 55
7.2. Informative References 57
Appendix A. Script to Generate IANA-Maintained YANG Modules 58
Acknowledgements 64
Contributors 64
Author's Address 64

1. Introduction

This document presents three IETF-defined YANG modules [RFC7950] and a script used to create four supporting IANA modules.

The three IETF modules are ietf-ssh-common (Section 2), ietf-ssh-client (Section 3), and ietf-ssh-server (Section 4). The "ietf-ssh-client" and "ietf-ssh-server" modules are the primary productions of this work, supporting the configuration and monitoring of SSH clients and servers.

The groupings defined in this document are expected to be used in conjunction with the groupings defined in an underlying transport-level module, such as the groupings defined in [RFC9643]. The transport-level data model enables the configuration of transport-level values, such as a remote address, a remote port, a local address, and a local port.

The four IANA modules are: iana-ssh-encryption-algs, iana-ssh-key-exchange-algs, iana-ssh-mac-algs, and iana-ssh-public-key-algs. These modules each define YANG enumerations providing support for an IANA-maintained algorithm registry.

This document assumes that the four IANA modules exist and presents a script in Appendix A that IANA may use to generate those YANG modules. This document does not publish the initial versions of these four modules. IANA publishes these modules.

1.1. Regarding the Three IETF Modules

The three IETF modules define features and groupings to model "generic" SSH clients and SSH servers, where "generic" should be interpreted as "least common denominator" rather than "complete." Support for the basic SSH protocol [RFC4252] [RFC4253] [RFC4254] is afforded by these modules, leaving configuration of advanced features (e.g., multiple channels) to augmentations made by consuming modules.

It is intended that the YANG groupings will be used by applications needing to configure SSH client and server protocol stacks. For instance, these groupings are used to help define the data models in [NETCONF-CLIENT-SERVER], for clients and servers using the Network Configuration Protocol (NETCONF) over SSH [RFC6242].

The "ietf-ssh-client" and "ietf-ssh-server" YANG modules each define one grouping, which is focused on just SSH-specific configuration, and specifically avoid any transport-level configuration, such as what ports to listen on or connect to. This affords applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the "ssh-server-grouping" grouping for the SSH parts it provides while adding data nodes for the TCP-level call-home configuration.

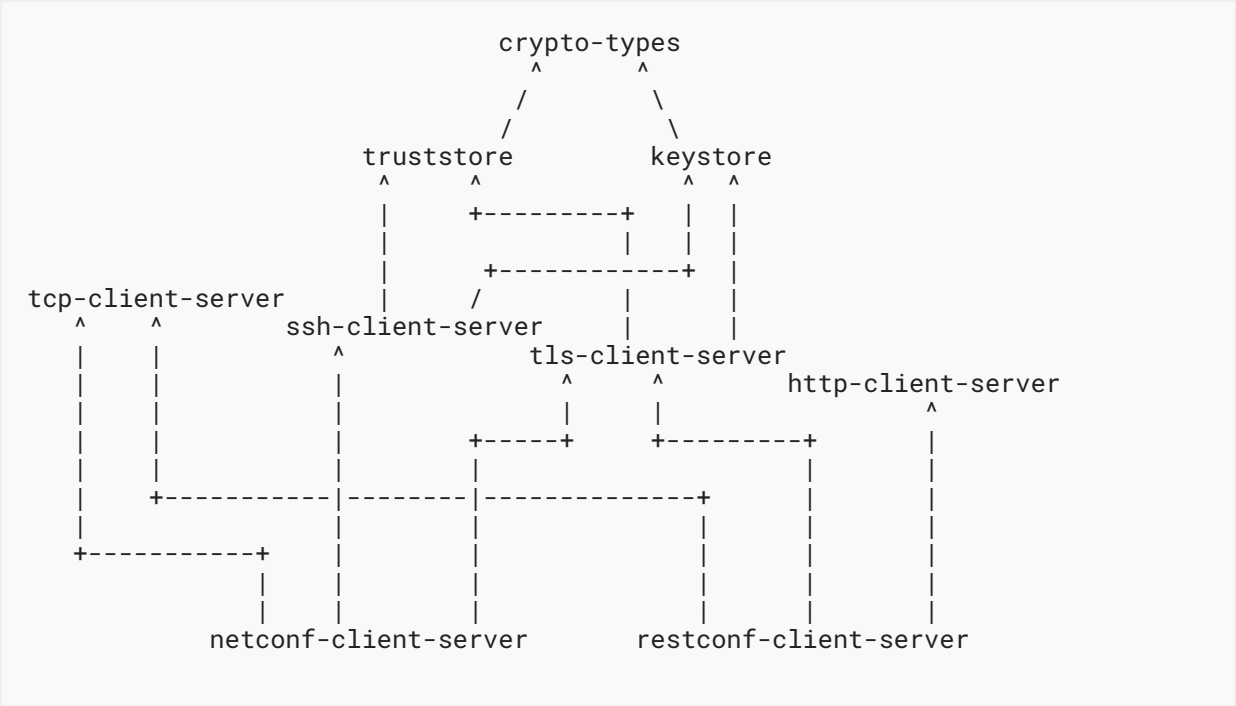
The modules defined in this document optionally support [RFC6187], which describes enabling host keys and public keys based on X.509v3 certificates.

1.2. Relation to Other RFCs

This document presents three YANG modules [RFC7950] that are part of a collection of RFCs that work together to ultimately support the configuration of both the clients and servers of both the NETCONF [RFC6241] and RESTCONF [RFC8040] protocols.

The dependency relationship between the primary YANG groupings defined in the various RFCs is presented in the below diagram. In some cases, a document may define secondary groupings that introduce dependencies not illustrated in the diagram. The labels in the diagram are shorthand names for the defining RFCs. The citation references for shorthand names are provided below the diagram.

Please note that the arrows in the diagram point from referencer to referenced. For example, the "crypto-types" RFC does not have any dependencies, whilst the "keystore" RFC depends on the "crypto-types" RFC.



Label in Diagram	Reference
crypto-types	[RFC9640]
truststore	[RFC9641]
keystore	[RFC9642]
tcp-client-server	[RFC9643]
ssh-client-server	RFC9644
tls-client-server	[RFC9645]
http-client-server	[HTTP-CLIENT-SERVER]
netconf-client-server	[NETCONF-CLIENT-SERVER]
restconf-client-server	[RESTCONF-CLIENT-SERVER]

Table 1: Label in Diagram to RFC Mapping

1.3. Specification Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.4. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, as described in [RFC9641] and [RFC9642], trust anchors and keys installed during manufacturing are expected to appear in <operational> (Section 5.3 of [RFC8342]) and <system> [SYSTEM-CONFIG] if implemented.

1.5. Conventions

Various examples in this document use "BASE64VALUE=" as a placeholder value for binary data that has been base64 encoded (per Section 9.8 of [RFC7950]). This placeholder value is used because real base64-encoded structures are often many lines long and hence distracting to the example being presented.

Various examples in this document use the XML [W3C.REC-xml-20081126] encoding. Other encodings, such as JSON [RFC8259], could alternatively be used.

Various examples in this document contain long lines that may be folded, as described in [RFC8792].

2. The "ietf-ssh-common" Module

The SSH common model presented in this section is common to both SSH clients and SSH servers. The "transport-params-grouping" grouping can be used to configure the list of SSH transport algorithms permitted by the SSH client or SSH server. The lists of permitted algorithms are in decreasing order of usage preference. The algorithm that appears first in the client list that also appears in the server list is the one that is used for the SSH transport layer connection. The ability to restrict the algorithms allowed is provided in this grouping for SSH clients and SSH servers that are capable of doing so and may serve to make SSH clients and SSH servers compliant with security policies.

2.1. Data Model Overview

This section provides an overview of the "ietf-ssh-common" module in terms of its features, identities, groupings, and protocol-accessible nodes.

2.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-ssh-common" module:

```
Features:
  +-- ssh-x509-certs
  +-- transport-params
  +-- asymmetric-key-pair-generation
  +-- algorithm-discovery
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Please refer to the YANG module for a description of each feature.

2.1.2. Groupings

The "ietf-ssh-common" module defines the following "grouping" statement:

- transport-params-grouping

This grouping is presented in the following subsection.

2.1.2.1. The "transport-params-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "transport-params-grouping" grouping:

```
grouping transport-params-grouping:
  +-- host-key
  |  +-- host-key-alg*   ssh-public-key-algorithm
  +-- key-exchange
  |  +-- key-exchange-alg*   ssh-key-exchange-algorithm
  +-- encryption
  |  +-- encryption-alg*   ssh-encryption-algorithm
  +-- mac
     +-- mac-alg*   ssh-mac-algorithm
```

Comments:

- This grouping is used by both the "ssh-client-grouping" and the "ssh-server-grouping" groupings defined in Sections 3.1.2.1 and 4.1.2.1, respectively.
- This grouping enables client and server configurations to specify the algorithms that are to be used when establishing SSH sessions.
- Each list is "ordered-by user".

2.1.3. Protocol-Accessible Nodes

The following tree diagram [RFC8340] lists all the protocol-accessible nodes defined in the "ietf-ssh-common" module without expanding the "grouping" statements:

```
module: ietf-ssh-common
  +--ro supported-algorithms {algorithm-discovery}?
     +--ro public-key-algorithms
     |  +--ro supported-algorithm*   ssh-public-key-algorithm
     +--ro encryption-algorithms
     |  +--ro supported-algorithm*   ssh-encryption-algorithm
     +--ro key-exchange-algorithms
     |  +--ro supported-algorithm*   ssh-key-exchange-algorithm
     +--ro mac-algorithms
        +--ro supported-algorithm*   ssh-mac-algorithm

  rpcs:
    +---x generate-asymmetric-key-pair
           {asymmetric-key-pair-generation}?
       +---w input
       |  +---w algorithm               ssh-public-key-algorithm
       |  +---w num-bits?               uint16
       |  +---w private-key-encoding
       |     +---w (private-key-encoding)
       |        +--:(cleartext) {ct:cleartext-private-keys}?
       |        |  +---w cleartext?   empty
       |        +--:(encrypted) {ct:encrypted-private-keys}?
       |        |  +---w encrypted
       |        |     +---w ks:encrypted-by-grouping
       |        +--:(hidden) {ct:hidden-private-keys}?
       |           +---w hidden?      empty
       +--ro output
          +--ro (key-or-hidden)?
             +--:(key)
             |  +---u ct:asymmetric-key-pair-grouping
             +--:(hidden)
                +--ro location?
                        instance-identifier
```

Comments:

- Protocol-accessible nodes are those nodes that are accessible when the module is "implemented", as described in Section 5.6.5 of [RFC7950].
- The protocol-accessible nodes for the "ietf-ssh-common" module are limited to the "supported-algorithms" container, which is constrained by the "algorithm-discovery" feature, and the "generate-asymmetric-key-pair" RPC, which is constrained by the "asymmetric-key-pair-generation" feature.
- The "encrypted-by-grouping" grouping is discussed in Section 2.1.3.1 of [RFC9642].
- The "asymmetric-key-pair-grouping" grouping is discussed in Section 2.1.4.6 of [RFC9640].

2.2. Example Usage

The following example illustrates the "transport-params-grouping' grouping when populated with some data.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<transport-params
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common">
  <host-key>
    <host-key-alg>x509v3-rsa2048-sha256</host-key-alg>
    <host-key-alg>ssh-rsa</host-key-alg>
    <host-key-alg>ssh-rsa@openssh.com</host-key-alg>
  </host-key>
  <key-exchange>
    <key-exchange-alg>diffie-hellman-group-exchange-sha256</key-exch\
ange-alg>
  </key-exchange>
  <encryption>
    <encryption-alg>aes256-ctr</encryption-alg>
    <encryption-alg>aes192-ctr</encryption-alg>
    <encryption-alg>aes128-ctr</encryption-alg>
    <encryption-alg>aes256-gcm@openssh.com</encryption-alg>
  </encryption>
  <mac>
    <mac-alg>hmac-sha2-256</mac-alg>
    <mac-alg>hmac-sha2-512</mac-alg>
  </mac>
</transport-params>
```

The following example illustrates operational state data indicating the SSH algorithms supported by the server.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<supported-algorithms
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common">
  <encryption-algorithms>
    <supported-algorithm>aes256-ctr</supported-algorithm>
    <supported-algorithm>arcfour256</supported-algorithm>
    <supported-algorithm>serpent256-ctr</supported-algorithm>
    <supported-algorithm>AEAD_AES_128_GCM</supported-algorithm>
    <supported-algorithm>AEAD_AES_256_GCM</supported-algorithm>
    <supported-algorithm>aes256-gcm@openssh.com</supported-algorithm>
  </encryption-algorithms>
  <key-exchange-algorithms>
    <supported-algorithm>ecdh-sha2-nistp256</supported-algorithm>
    <supported-algorithm>rsa2048-sha256</supported-algorithm>
    <supported-algorithm>gss-group14-sha1-nistp256</supported-algori\
thm>
    <supported-algorithm>gss-gex-sha1-nistp256</supported-algorithm>
    <supported-algorithm>gss-group14-sha256-1.2.840.10045.3.1.1</sup\
ported-algorithm>
    <supported-algorithm>curve25519-sha256</supported-algorithm>
  </key-exchange-algorithms>
  <mac-algorithms>
    <supported-algorithm>hmac-sha2-256</supported-algorithm>
    <supported-algorithm>hmac-sha2-512</supported-algorithm>
    <supported-algorithm>AEAD_AES_256_GCM</supported-algorithm>
  </mac-algorithms>
  <public-key-algorithms>
    <supported-algorithm>rsa-sha2-256</supported-algorithm>
    <supported-algorithm>rsa-sha2-512</supported-algorithm>
    <supported-algorithm>spki-sign-rsa</supported-algorithm>
    <supported-algorithm>pgp-sign-dss</supported-algorithm>
    <supported-algorithm>x509v3-rsa2048-sha256</supported-algorithm>
    <supported-algorithm>ecdsa-sha2-nistp256</supported-algorithm>
    <supported-algorithm>ecdsa-sha2-1.3.132.0.37</supported-algorith\
m>
    <supported-algorithm>ssh-ed25519</supported-algorithm>
    <supported-algorithm>ssh-rsa@openssh.com</supported-algorithm>
  </public-key-algorithms>
</supported-algorithms>
```

The following example illustrates the "generate-asymmetric-key-pair" RPC.

REQUEST

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <generate-asymmetric-key-pair
    xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common">
    <algorithm>ecdsa-sha2-nistp256</algorithm>
    <num-bits>521</num-bits>
    <private-key-encoding>
      <encrypted>
        <asymmetric-key-ref>hidden-asymmetric-key</asymmetric-key-re\
f>
      </encrypted>
    </private-key-encoding>
  </generate-asymmetric-key-pair>
</rpc>
```

RESPONSE

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types"
  xmlns:sshcmn="urn:ietf:params:xml:ns:yang:ietf-ssh-common">
  <sshcmn:public-key-format>ct:subject-public-key-info-format</sshcm\
n:public-key-format>
  <sshcmn:public-key>BASE64VALUE=</sshcmn:public-key>
  <sshcmn:private-key-format>ct:ec-private-key-format</sshcmn:privat\
e-key-format>
  <sshcmn:cleartext-private-key>BASE64VALUE=</sshcmn:cleartext-priva\
te-key>
</rpc-reply>
```

2.3. YANG Module

This YANG module has normative references to [RFC4250], [RFC4253], [RFC6187], and [FIPS_186-5].

```
<CODE BEGINS> file "ietf-ssh-common@2024-03-16.yang"

module ietf-ssh-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-common";
  prefix sshcmn;

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC 9640: YANG Data Types and Groupings for Cryptography";
```

```
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC 9642: A YANG Data Model for a Keystore";
  }

  import iana-ssh-encryption-algs {
    prefix sshea;
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  import iana-ssh-key-exchange-algs {
    prefix sshkea;
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  import iana-ssh-mac-algs {
    prefix sshma;
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  import iana-ssh-public-key-algs {
    prefix sshpka;
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

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

  contact
    "WG Web:   https://datatracker.ietf.org/wg/netconf
     WG List:  NETCONF WG list <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kent+ietf@watsen.net>
     Author:   Gary Wu <mailto:garywu@cisco.com>";

  description
    "This module defines common features and groupings for
     Secure Shell (SSH).

     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.

     Copyright (c) 2024 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 Revised
```

```
      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 9644
      (https://www.rfc-editor.org/info/rfc9644); see the RFC
      itself for full legal notices.";

    revision 2024-03-16 {
      description
        "Initial version.";
      reference
        "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
    }

    // Features

    feature ssh-x509-certs {
      description
        "X.509v3 certificates are supported for SSH.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
                   Authentication";
    }

    feature transport-params {
      description
        "SSH transport layer parameters are configurable.";
    }

    feature asymmetric-key-pair-generation {
      description
        "Indicates that the server implements the
         'generate-asymmetric-key-pair' RPC.";
    }

    feature algorithm-discovery {
      description
        "Indicates that the server implements the
         'supported-algorithms' container.";
    }

    // Typedefs

    typedef ssh-public-key-algorithm {
      type union {
        type sshpka:ssh-public-key-algorithm;
        type string {
          length "1..64" {
            description
              "Non-IANA-maintained algorithms must include the
               at sign (@) in them, per Section 4.6.1 of RFC
               4250.";
            reference
              "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                         Numbers";
          }
          pattern '.*@.*' {
```

```
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC
             4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                       Numbers";
        }
      }
    }
    description
      "A type that enables the public key algorithm to be
       either an IANA-maintained public key algorithm in
       the 'iana-ssh-public-key-algs' YANG module (RFC 9644)
       or a locally defined algorithm, per Section 4.6.1
       of RFC 4250.";
    reference
      "RFC 4250: The Secure Shell (SSH) Protocol Assigned Numbers
       RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  typedef ssh-key-exchange-algorithm {
    type union {
      type sshkea:ssh-key-exchange-algorithm;
      type string {
        length "1..64" {
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC 4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                       Numbers";
        }
        pattern '.*@.*' {
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC 4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                     Numbers";
        }
      }
    }
    description
      "A type that enables the key exchange algorithm to be
       either an IANA-maintained key exchange algorithm in
       the 'iana-ssh-key-exchange-algs' YANG module (RFC 9644)
       or a locally defined algorithm, per Section 4.6.1
       of RFC 4250.";
    reference
      "RFC 4250: The Secure Shell (SSH) Protocol Assigned Numbers
       RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  typedef ssh-encryption-algorithm {
    type union {
      type sshea:ssh-encryption-algorithm;
      type string {
```

```
        length "1..64" {
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC
             4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                       Numbers";
        }
        pattern '.*@.*' {
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC
             4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                       Numbers";
        }
      }
    }
    description
      "A type that enables the encryption algorithm to be
       either an IANA-maintained encryption algorithm in
       the 'iana-ssh-encryption-algs' YANG module (RFC 9644)
       or a locally defined algorithm, per Section 4.6.1
       of RFC 4250.";
    reference
      "RFC 4250: The Secure Shell (SSH) Protocol Assigned Numbers
       RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  typedef ssh-mac-algorithm {
    type union {
      type sshma:ssh-mac-algorithm;
      type string {
        length "1..64" {
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC
             4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                       Numbers";
        }
        pattern '.*@.*' {
          description
            "Non-IANA-maintained algorithms must include the
             at sign (@) in them, per Section 4.6.1 of RFC
             4250.";
          reference
            "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                       Numbers";
        }
      }
    }
    description
      "A type that enables the message authentication code (MAC)
       algorithm to be either an IANA-maintained MAC algorithm
```

```
       in the 'iana-ssh-mac-algs' YANG module (RFC 9644)
       or a locally defined algorithm, per Section 4.6.1
       of RFC 4250.";
    reference
      "RFC 4250: The Secure Shell (SSH) Protocol Assigned Numbers
       RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  // Groupings

  grouping transport-params-grouping {
    description
      "A reusable grouping for SSH transport parameters.";
    reference
      "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    container host-key {
      description
        "Parameters regarding host key.";
      leaf-list host-key-alg {
        type ssh-public-key-algorithm;
        ordered-by user;
        description
          "Acceptable host key algorithms in order of decreasing
           preference.

           If this leaf-list is not configured (has zero
           elements), the acceptable host key algorithms are
           implementation-defined.";
        reference
          "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
      }
    }
    container key-exchange {
      description
        "Parameters regarding key exchange.";
      leaf-list key-exchange-alg {
        type ssh-key-exchange-algorithm;
        ordered-by user;
        description
          "Acceptable key exchange algorithms in order of decreasing
           preference.

           If this leaf-list is not configured (has zero
           elements), the acceptable key exchange algorithms are
           implementation-defined.";
      }
    }
    container encryption {
      description
        "Parameters regarding encryption.";
      leaf-list encryption-alg {
        type ssh-encryption-algorithm;
        ordered-by user;
        description
          "Acceptable encryption algorithms in order of decreasing
           preference.

           If this leaf-list is not configured (has zero
```

```
            elements), the acceptable encryption algorithms are
            implementation-defined.";
      }
    }
    container mac {
      description
        "Parameters regarding message authentication code (MAC).";
      leaf-list mac-alg {
        type ssh-mac-algorithm;
        ordered-by user;
        description
          "Acceptable MAC algorithms in order of decreasing
           preference.

           If this leaf-list is not configured (has zero
           elements), the acceptable MAC algorithms are
           implementation-defined.";
      }
    }
  }

  // Protocol-accessible Nodes

  container supported-algorithms {
    if-feature "algorithm-discovery";
    config false;
    description
      "Identifies all of the supported algorithms.";
    container public-key-algorithms {
      description
        "A container for a list of public key algorithms
         supported by the server.";
      leaf-list supported-algorithm {
        type ssh-public-key-algorithm;
        description
          "A public key algorithm supported by the server.";
      }
    }
    container encryption-algorithms {
      description
        "A container for a list of encryption algorithms
         supported by the server.";
      leaf-list supported-algorithm {
        type ssh-encryption-algorithm;
        description
          "An encryption algorithm supported by the server.";
      }
    }
    container key-exchange-algorithms {
      config false;
      description
        "A container for a list of key exchange algorithms
         supported by the server.";
      leaf-list supported-algorithm {
        type ssh-key-exchange-algorithm;
        description
          "A key exchange algorithm supported by the server.";
      }
```

```
    }
    container mac-algorithms {
      config false;
      description
        "A container for a list of MAC algorithms
         supported by the server.";
      leaf-list supported-algorithm {
        type ssh-mac-algorithm;
        description
          "A MAC algorithm supported by the server.";
      }
    }
  }

  rpc generate-asymmetric-key-pair {
    if-feature "asymmetric-key-pair-generation";
    description
      "Requests the device to generate a public key using
       the specified key algorithm.";
    input {
      leaf algorithm {
        type ssh-public-key-algorithm;
        mandatory true;
        description
          "The algorithm to be used when generating the key.";
      }
      leaf num-bits {
        type uint16;
        description
          "Specifies the number of bits in the key to create.
           For RSA keys, the minimum size is 1024 bits and
           the default is 3072 bits.  Generally, 3072 bits is
           considered sufficient.  DSA keys must be exactly 1024
           bits, as specified by FIPS 186-5.  For Elliptic Curve
           Digital Signature Algorithm (ECDSA) keys, the
           'num-bits' value determines the key length by selecting
           from one of three elliptic curve sizes: 256, 384, or
           521 bits.  Attempting to use bit lengths other than
           these three values for ECDSA keys will fail.  ECDSA-SK,
           Ed25519, and Ed25519-SK keys have a fixed length, and
           thus, the 'num-bits' value is not specified.";
        reference
          "FIPS 186-5: Digital Signature Standard (DSS)";
      }
      container private-key-encoding {
        description
          "Indicates how the private key is to be encoded.";
        choice private-key-encoding {
          mandatory true;
          description
            "A choice amongst optional private key handling.";
          case cleartext {
            if-feature "ct:cleartext-private-keys";
            leaf cleartext {
              type empty;
              description
                "Indicates that the private key is to be returned
                 as a cleartext value.";
```

```
            }
          }
          case encrypted {
            if-feature "ct:encrypted-private-keys";
            container encrypted {
              description
                "Indicates that the private key is to be encrypted
                 using the specified symmetric or asymmetric key.";
              uses ks:encrypted-by-grouping;
            }
          }
          case hidden {
            if-feature "ct:hidden-private-keys";
            leaf hidden {
              type empty;
              description
                "Indicates that the private key is to be hidden.

                 Unlike the 'cleartext' and 'encrypt' options, the
                 key returned is a placeholder for an internally
                 stored key.  See the 'Support for Built-in Keys'
                 section in RFC 9642 for information about hidden
                 keys.

                 It is expected that the server will instantiate
                 the hidden key in the same location where built-in
                 keys are located.  Rather than returning the key,
                 just the key's location is returned in the output.";
            }
          }
        }
      }
    }
    output {
      choice key-or-hidden {
        case key {
          uses ct:asymmetric-key-pair-grouping;
        }
        case hidden {
          leaf location {
            type instance-identifier;
            description
              "The location to where a hidden key was created.";
          }
        }
        description
          "The output can be either a key (for cleartext and
           encrypted keys) or the location to where the key
           was created (for hidden keys).";
      }
    }
  } // end generate-asymmetric-key-pair

}

<CODE ENDS>
```

3. The "ietf-ssh-client" Module

This section defines a YANG 1.1 [RFC7950] module called "ietf-ssh-client". A high-level overview of the module is provided in Section 3.1. Examples illustrating the module's use are provided in Section 3.2 ("Example Usage"). The YANG module itself is defined in Section 3.3.

3.1. Data Model Overview

This section provides an overview of the "ietf-ssh-client" module in terms of its features and groupings.

3.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-ssh-client" module:

```
Features:
  +-- ssh-client-keepalives
  +-- client-ident-password
  +-- client-ident-publickey
  +-- client-ident-hostbased
  +-- client-ident-none
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Please refer to the YANG module for a description of each feature.

3.1.2. Groupings

The "ietf-ssh-client" module defines the following "grouping" statement:

- ssh-client-grouping

This grouping is presented in the following subsection.

3.1.2.1. The "ssh-client-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "ssh-client-grouping" grouping:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

  grouping ssh-client-grouping:
    +-- client-identity
    |  +-- username?      string
    |  +-- public-key! {client-ident-publickey}?
    |  |  +---u ks:inline-or-keystore-asymmetric-key-grouping
    |  +-- password! {client-ident-password}?
    |  |  +---u ct:password-grouping
    |  +-- hostbased! {client-ident-hostbased}?
    |  |  +---u ks:inline-or-keystore-asymmetric-key-grouping
    |  +-- none?          empty {client-ident-none}?
    |  +-- certificate! {sshcmn:ssh-x509-certs}?
    |     +---u ks:inline-or-keystore-end-entity-cert-with-key-group\
ing
    +-- server-authentication
    |  +-- ssh-host-keys!
    |  |  +---u ts:inline-or-truststore-public-keys-grouping
    |  +-- ca-certs! {sshcmn:ssh-x509-certs}?
    |  |  +---u ts:inline-or-truststore-certs-grouping
    |  +-- ee-certs! {sshcmn:ssh-x509-certs}?
    |     +---u ts:inline-or-truststore-certs-grouping
    +-- transport-params {sshcmn:transport-params}?
    |  +---u sshcmn:transport-params-grouping
    +-- keepalives! {ssh-client-keepalives}?
       +-- max-wait?       uint16
       +-- max-attempts?   uint8
```

Comments:

- The "client-identity" node configures a "username" and authentication methods, each enabled by a "feature" statement defined in Section 3.1.1.
- The "server-authentication" node configures trust anchors for authenticating the SSH server, with each option enabled by a "feature" statement.
- The "transport-params" node, which must be enabled by a feature, configures parameters for the SSH sessions established by this configuration.
- The "keepalives" node, which must be enabled by a feature, configures a "presence" container for testing the aliveness of the SSH server. The aliveness-test occurs at the SSH protocol layer.
- For the referenced grouping statements:
  - The "inline-or-keystore-asymmetric-key-grouping" grouping is discussed in Section 2.1.3.4 of [RFC9642].
  - The "inline-or-keystore-end-entity-cert-with-key-grouping" grouping is discussed in Section 2.1.3.6 of [RFC9642].
  - The "inline-or-truststore-public-keys-grouping" grouping is discussed in Section 2.1.3.4 of [RFC9641].
  - The "inline-or-truststore-certs-grouping" grouping is discussed in Section 2.1.3.3 of [RFC9641].
  - The "transport-params-grouping" grouping is discussed in Section 2.1.2.1 in this document.

3.1.3. Protocol-Accessible Nodes

The "ietf-ssh-client" module defines only "grouping" statements that are used by other modules to instantiate protocol-accessible nodes. Thus, this module, when implemented, does not itself define any protocol-accessible nodes.

3.2. Example Usage

This section presents two examples showing the "ssh-client-grouping" grouping populated with some data. These examples are effectively the same, except the first configures the client identity using an inlined key, while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 2.2.1 of [RFC9641] and Section 2.2.1 of [RFC9642].

The following configuration example uses inline-definitions for the client identity and server authentication:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<ssh-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client"
  xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <username>foobar</username>
    <public-key>
      <inline-definition>
        <private-key-format>ct:rsa-private-key-format</private-key-f\
ormat>
        <cleartext-private-key>BASE64VALUE=</cleartext-private-key>
      </inline-definition>
    </public-key>
  </client-identity>

  <!-- which host keys will this client trust -->
  <server-authentication>
    <ssh-host-keys>
      <inline-definition>
        <public-key>
          <name>corp-fw1</name>
          <public-key-format>ct:ssh-public-key-format</public-key-fo\
rmat>
          <public-key>BASE64VALUE=</public-key>
        </public-key>
        <public-key>
          <name>corp-fw2</name>
          <public-key-format>ct:ssh-public-key-format</public-key-fo\
rmat>
          <public-key>BASE64VALUE=</public-key>
```

```
          </public-key>
        </inline-definition>
      </ssh-host-keys>
      <ca-certs>
        <inline-definition>
          <certificate>
            <name>Server Cert Issuer #1</name>
            <cert-data>BASE64VALUE=</cert-data>
          </certificate>
          <certificate>
            <name>Server Cert Issuer #2</name>
            <cert-data>BASE64VALUE=</cert-data>
          </certificate>
        </inline-definition>
      </ca-certs>
      <ee-certs>
        <inline-definition>
          <certificate>
            <name>My Application #1</name>
            <cert-data>BASE64VALUE=</cert-data>
          </certificate>
          <certificate>
            <name>My Application #2</name>
            <cert-data>BASE64VALUE=</cert-data>
          </certificate>
        </inline-definition>
      </ee-certs>
    </server-authentication>

    <keepalives>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </keepalives>

  </ssh-client>
```

The following configuration example uses central-keystore-references for the client identity and central-truststore-references for server authentication from the keystore:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<ssh-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client"
  xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <username>foobar</username>
    <public-key>
      <central-keystore-reference>ssh-rsa-key</central-keystore-refe\
rence>
    </public-key>
    <certificate>
      <central-keystore-reference>
        <asymmetric-key>ssh-rsa-key-with-cert</asymmetric-key>
        <certificate>ex-rsa-cert2</certificate>
      </central-keystore-reference>
    </certificate>
  </client-identity>

  <!-- which host-keys will this client trust -->
  <server-authentication>
    <ssh-host-keys>
      <central-truststore-reference>trusted-ssh-public-keys</central\
-truststore-reference>
    </ssh-host-keys>
    <ca-certs>
      <central-truststore-reference>trusted-server-ca-certs</central\
-truststore-reference>
    </ca-certs>
    <ee-certs>
      <central-truststore-reference>trusted-server-ee-certs</central\
-truststore-reference>
    </ee-certs>
  </server-authentication>

  <keepalives>
    <max-wait>30</max-wait>
    <max-attempts>3</max-attempts>
  </keepalives>

</ssh-client>
```

3.3. YANG Module

This YANG module has normative references to [RFC4252], [RFC4254], [RFC8341], [RFC9640], [RFC9641], and [RFC9642].

```
<CODE BEGINS> file "ietf-ssh-client@2024-03-16.yang"
```

```
module ietf-ssh-client {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-client";
  prefix sshc;

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

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC 9640: YANG Data Types and Groupings for Cryptography";
  }

  import ietf-truststore {
    prefix ts;
    reference
      "RFC 9641: A YANG Data Model for a Truststore";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC 9642: A YANG Data Model for a Keystore";
  }

  import ietf-ssh-common {
    prefix sshcmn;
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

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

  contact
    "WG Web:   https://datatracker.ietf.org/wg/netconf
     WG List:  NETCONF WG list <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kent+ietf@watsen.net>";

  description
    "This module defines a reusable grouping for SSH clients that
     can be used as a basis for specific SSH client instances.

     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.

     Copyright (c) 2024 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 Revised
     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 9644
     (https://www.rfc-editor.org/info/rfc9644); see the RFC
     itself for full legal notices.";

   revision 2024-03-16 {
     description
       "Initial version.";
     reference
       "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
   }

   // Features

   feature ssh-client-keepalives {
     description
       "SSH keepalive parameters are configurable for
        SSH clients on the server implementing this feature.";
   }

   feature client-ident-publickey {
     description
       "Indicates that the 'publickey' authentication type, per
        RFC 4252, is supported for client identification.
        The 'publickey' authentication type is required by
        RFC 4252, but common implementations allow it to
        be disabled.";
     reference
       "RFC 4252:
         The Secure Shell (SSH) Authentication Protocol";
   }

   feature client-ident-password {
     description
       "Indicates that the 'password' authentication type, per
        RFC 4252, is supported for client identification.";
     reference
       "RFC 4252:
         The Secure Shell (SSH) Authentication Protocol";
   }

   feature client-ident-hostbased {
     description
       "Indicates that the 'hostbased' authentication type, per
        RFC 4252, is supported for client identification.";
     reference
       "RFC 4252:
         The Secure Shell (SSH) Authentication Protocol";
   }

   feature client-ident-none {
     description
       "Indicates that the 'none' authentication type, per
```

```
       RFC 4252, is supported for client identification.
       It is NOT RECOMMENDED to enable this feature.";
    reference
      "RFC 4252:
       The Secure Shell (SSH) Authentication Protocol";
  }

  // Groupings

  grouping ssh-client-grouping {
    description
      "A reusable grouping for configuring an SSH client without
       any consideration for how an underlying TCP session is
       established.

       Note that this grouping uses fairly typical descendant
       node names such that a nesting of 'uses' statements will
       have name conflicts.  It is intended that the consuming
       data model will resolve the issue (e.g., by wrapping
       the 'uses' statement in a container called
       'ssh-client-parameters').  This model purposely does
       not do this itself so as to provide maximum flexibility
       to consuming models.";

    container client-identity {
      nacm:default-deny-write;
      description
        "The username and authentication methods for the client.
         The authentication methods are unordered.  Clients may
         initially send any configured method or, per Section 5.2 of
         RFC 4252, send the 'none' method to prompt the server
         to provide a list of productive methods.  Whenever a
         choice amongst methods arises, implementations SHOULD
         use a default ordering that prioritizes automation
         over human interaction.";
      leaf username {
        type string;
        description
          "The username of this user.  This will be the username
           used, for instance, to log into an SSH server.";
      }
      container public-key {
        if-feature "client-ident-publickey";
        presence
          "Indicates that public-key-based authentication has been
           configured.  This statement is present so the mandatory
           descendant nodes do not imply that this node must be
           configured.";
        description
          "A locally defined or referenced asymmetric key
           pair to be used for client identification.";
        reference
          "RFC 9642: A YANG Data Model for a Keystore";
        uses ks:inline-or-keystore-asymmetric-key-grouping {
          refine "inline-or-keystore/inline/inline-definition" {
            must 'not(public-key-format) or derived-from-or-self'
                + '(public-key-format, "ct:ssh-public-key-format")';
          }
```

```
          refine "inline-or-keystore/central-keystore/"
               + "central-keystore-reference" {
            must 'not(deref(.)/../ks:public-key-format) or derived-'
               + 'from-or-self(deref(.)/../ks:public-key-format, '
               + '"ct:ssh-public-key-format")';
          }
        }
      }
      container password {
        if-feature "client-ident-password";
        presence
          "Indicates that password-based authentication has been
           configured.  This statement is present so the mandatory
           descendant nodes do not imply that this node must be
           configured.";
        description
          "A password to be used to authenticate the client's
           identity.";
        uses ct:password-grouping;
      }
      container hostbased {
        if-feature "client-ident-hostbased";
        presence
          "Indicates that host-based authentication is configured.
           This statement is present so the mandatory descendant
           nodes do not imply that this node must be configured.";
        description
          "A locally defined or referenced asymmetric key
           pair to be used for host identification.";
        reference
          "RFC 9642: A YANG Data Model for a Keystore";
        uses ks:inline-or-keystore-asymmetric-key-grouping {
          refine "inline-or-keystore/inline/inline-definition" {
            must 'not(public-key-format) or derived-from-or-self('
               + 'public-key-format, "ct:ssh-public-key-format")';
          }
          refine "inline-or-keystore/central-keystore/"
               + "central-keystore-reference" {
            must 'not(deref(.)/../ks:public-key-format) or derived-'
               + 'from-or-self(deref(.)/../ks:public-key-format, '
               + '"ct:ssh-public-key-format")';
          }
        }
      }
      leaf none {
        if-feature "client-ident-none";
        type empty;
        description
          "Indicates that the 'none' algorithm is used for client
           identification.";
      }
      container certificate {
        if-feature "sshcmn:ssh-x509-certs";
        presence
          "Indicates that certificate-based authentication has been
           configured.  This statement is present so the mandatory
           descendant nodes do not imply that this node must be
           configured.";
```

```
        description
          "A locally defined or referenced certificate
           to be used for client identification.";
        reference
          "RFC 9642: A YANG Data Model for a Keystore";
        uses
          ks:inline-or-keystore-end-entity-cert-with-key-grouping {
          refine "inline-or-keystore/inline/inline-definition" {
            must 'not(public-key-format) or derived-from-or-self('
               + 'public-key-format, "ct:subject-public-key-info-'
               + 'format")';
          }
          refine "inline-or-keystore/central-keystore/"
               + "central-keystore-reference/asymmetric-key" {
            must 'not(deref(.)/../ks:public-key-format) or derived-'
               + 'from-or-self(deref(.)/../ks:public-key-format, '
               + '"ct:subject-public-key-info-format")';
          }
        }
      }
    } // container client-identity

    container server-authentication {
      nacm:default-deny-write;
      must 'ssh-host-keys or ca-certs or ee-certs';
      description
        "Specifies how the SSH client can authenticate SSH servers.
         Any combination of authentication methods is additive and
         unordered.";
      container ssh-host-keys {
        presence
          "Indicates that the SSH host key have been configured.
           This statement is present so the mandatory descendant
           nodes do not imply that this node must be configured.";
        description
          "A bag of SSH host keys used by the SSH client to
           authenticate SSH server host keys.  A server host key
           is authenticated if it is an exact match to a
           configured SSH host key.";
        reference
          "RFC 9641: A YANG Data Model for a Truststore";
        uses ts:inline-or-truststore-public-keys-grouping {
          refine
            "inline-or-truststore/inline/inline-definition/public"
            + "-key" {
            must 'derived-from-or-self(public-key-format,'
               + ' "ct:ssh-public-key-format")';
          }
          refine "inline-or-truststore/central-truststore/"
               + "central-truststore-reference" {
            must 'not(deref(.)/../ts:public-key/ts:public-key-'
               + 'format[not(derived-from-or-self(., "ct:ssh-'
               + 'public-key-format"))])';
          }
        }
      }
      container ca-certs {
        if-feature "sshcmn:ssh-x509-certs";
```

```
        presence
          "Indicates that the CA certificates have been configured.
           This statement is present so the mandatory descendant
           nodes do not imply that this node must be configured.";
        description
          "A set of Certification Authority (CA) certificates used by
           the SSH client to authenticate SSH servers.  A server
           is authenticated if its certificate has a valid chain
           of trust to a configured CA certificate.";
        reference
          "RFC 9641: A YANG Data Model for a Truststore";
        uses ts:inline-or-truststore-certs-grouping;
      }
      container ee-certs {
        if-feature "sshcmn:ssh-x509-certs";
        presence
          "Indicates that the EE certificates have been configured.
           This statement is present so the mandatory descendant
           nodes do not imply that this node must be configured.";
        description
          "A set of end-entity (EE) certificates used by the SSH
           client to authenticate SSH servers.  A server is
           authenticated if its certificate is an exact match to a
           configured end-entity certificate.";
        reference
          "RFC 9641: A YANG Data Model for a Truststore";
        uses ts:inline-or-truststore-certs-grouping;
      }
    } // container server-authentication

    container transport-params {
      nacm:default-deny-write;
      if-feature "sshcmn:transport-params";
      description
        "Configurable parameters of the SSH transport layer.";
      uses sshcmn:transport-params-grouping;
    } // container transport-parameters

    container keepalives {
      nacm:default-deny-write;
      if-feature "ssh-client-keepalives";
      presence
        "Indicates that the SSH client proactively tests the
         aliveness of the remote SSH server.";
      description
        "Configures the keepalive policy to proactively test
         the aliveness of the SSH server.  An unresponsive SSH
         server is dropped after approximately max-wait *
         max-attempts seconds.  Per Section 4 of RFC 4254,
         the SSH client SHOULD send an SSH_MSG_GLOBAL_REQUEST
         message with a purposely nonexistent 'request name'
         value (e.g., keepalive@example.com) and the 'want reply'
         value set to '1'.";
      reference
        "RFC 4254: The Secure Shell (SSH) Connection Protocol";
      leaf max-wait {
        type uint16 {
          range "1..max";
```

```
        }
        units "seconds";
        default "30";
        description
          "Sets the amount of time in seconds after which an
           SSH-level message will be sent to test the aliveness
           of the SSH server if no data has been received from the
           SSH server.";
      }
      leaf max-attempts {
        type uint8;
        default "3";
        description
          "Sets the maximum number of sequential keepalive
           messages that can fail to obtain a response from
           the SSH server before assuming the SSH server is
           no longer alive.";
      }
    } // container keepalives
  } // grouping ssh-client-grouping

}

<CODE ENDS>
```

4. The "ietf-ssh-server" Module

This section defines a YANG 1.1 module called "ietf-ssh-server". A high-level overview of the module is provided in Section 4.1. Examples illustrating the module's use are provided in Section 4.2 ("Example Usage"). The YANG module itself is defined in Section 4.3.

4.1. Data Model Overview

This section provides an overview of the "ietf-ssh-server" module in terms of its features and groupings.

4.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-ssh-server" module:

```
Features:
  +-- ssh-server-keepalives
  +-- local-users-supported
  +-- local-user-auth-publickey {local-users-supported}?
  +-- local-user-auth-password {local-users-supported}?
  +-- local-user-auth-hostbased {local-users-supported}?
  +-- local-user-auth-none {local-users-supported}?
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

Please refer to the YANG module for a description of each feature.

4.1.2. Groupings

The "ietf-ssh-server" module defines the following "grouping" statement:

- ssh-server-grouping

This grouping is presented in the following subsection.

4.1.2.1. The "ssh-server-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "ssh-server-grouping" grouping:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

  grouping ssh-server-grouping:
    +-- server-identity
    |  +-- host-key* [name]
    |     +-- name                string
    |     +-- (host-key-type)
    |        +--:(public-key)
    |        |  +-- public-key
    |        |     +---u ks:inline-or-keystore-asymmetric-key-groupi\
ng
    |        +--:(certificate)
    |           +-- certificate {sshcmn:ssh-x509-certs}?
    |              +---u ks:inline-or-keystore-end-entity-cert-with-\
key-grouping
    +-- client-authentication
    |  +-- users {local-users-supported}?
    |  |  +-- user* [name]
    |  |     +-- name          string
    |  |     +-- public-keys! {local-user-auth-publickey}?
    |  |     |  +---u ts:inline-or-truststore-public-keys-grouping
    |  |     +-- password
    |  |     |  +-- hashed-password?   ianach:crypt-hash
    |  |     |  |       {local-user-auth-password}?
    |  |     |  +--ro last-modified?     yang:date-and-time
    |  |     +-- hostbased! {local-user-auth-hostbased}?
    |  |     |  +---u ts:inline-or-truststore-public-keys-grouping
    |  |     +-- none?          empty {local-user-auth-none}?
    |  +-- ca-certs! {sshcmn:ssh-x509-certs}?
    |  |  +---u ts:inline-or-truststore-certs-grouping
    |  +-- ee-certs! {sshcmn:ssh-x509-certs}?
    |     +---u ts:inline-or-truststore-certs-grouping
    +-- transport-params {sshcmn:transport-params}?
    |  +---u sshcmn:transport-params-grouping
    +-- keepalives! {ssh-server-keepalives}?
       +-- max-wait?       uint16
       +-- max-attempts?   uint8
```

Comments:

- The "server-identity" node configures the authentication methods the server can use to identify itself to clients. The ability to use a certificate is enabled by a "feature".

- The "client-authentication" node configures trust anchors for authenticating the SSH client, with each option enabled by a "feature" statement.
- The "transport-params" node, which must be enabled by a feature, configures parameters for the SSH sessions established by this configuration.
- The "keepalives" node, which must be enabled by a feature, configures a "presence" container for testing the aliveness of the SSH client. The aliveness-test occurs at the SSH protocol layer.
- For the referenced grouping statements:
  - The "inline-or-keystore-asymmetric-key-grouping" grouping is discussed in Section 2.1.3.4 of [RFC9642].
  - The "inline-or-keystore-end-entity-cert-with-key-grouping" grouping is discussed in Section 2.1.3.6 of [RFC9642].
  - The "inline-or-truststore-public-keys-grouping" grouping is discussed in Section 2.1.3.4 of [RFC9641].
  - The "inline-or-truststore-certs-grouping" grouping is discussed in Section 2.1.3.3 of [RFC9641].
  - The "transport-params-grouping" grouping is discussed in Section 2.1.2.1 in this document.

4.1.3. Protocol-Accessible Nodes

The "ietf-ssh-server" module defines only "grouping" statements that are used by other modules to instantiate protocol-accessible nodes. Thus, this module, when implemented, does not itself define any protocol-accessible nodes.

4.2. Example Usage

This section presents two examples showing the "ssh-server-grouping" grouping populated with some data. These examples are effectively the same, except the first configures the server identity using an inlined key, while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 2.2.1 of [RFC9641] and Section 2.2.1 of [RFC9642].

The following configuration example uses inline-definitions for the server identity and client authentication:

```
================ NOTE: '\' line wrapping per RFC 8792 ================

<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<ssh-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server"
  xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">

  <!-- the host-key this SSH server will present -->
  <server-identity>
    <host-key>
      <name>my-pubkey-based-host-key</name>
```

```
      <public-key>
        <inline-definition>
          <private-key-format>ct:rsa-private-key-format</private-key\
-format>
          <cleartext-private-key>BASE64VALUE=</cleartext-private-key>
        </inline-definition>
      </public-key>
    </host-key>
    <host-key>
      <name>my-cert-based-host-key</name>
      <certificate>
        <inline-definition>
          <private-key-format>ct:rsa-private-key-format</private-key\
-format>
          <cleartext-private-key>BASE64VALUE=</cleartext-private-key>
          <cert-data>BASE64VALUE=</cert-data>
        </inline-definition>
      </certificate>
    </host-key>
  </server-identity>

  <!-- the client credentials this SSH server will trust -->
  <client-authentication>
    <users>
      <user>
        <name>mary</name>
        <password>
          <hashed-password>$0$example-secret</hashed-password>
        </password>
        <public-keys>
          <inline-definition>
            <public-key>
              <name>Mary-Key-1</name>
              <public-key-format>ct:ssh-public-key-format</public-ke\
y-format>
              <public-key>BASE64VALUE=</public-key>
            </public-key>
            <public-key>
              <name>Mary-Key-2</name>
              <public-key-format>ct:ssh-public-key-format</public-ke\
y-format>
              <public-key>BASE64VALUE=</public-key>
            </public-key>
          </inline-definition>
        </public-keys>
      </user>
    </users>
    <ca-certs>
      <inline-definition>
        <certificate>
          <name>Identity Cert Issuer #1</name>
          <cert-data>BASE64VALUE=</cert-data>
        </certificate>
        <certificate>
          <name>Identity Cert Issuer #2</name>
          <cert-data>BASE64VALUE=</cert-data>
        </certificate>
      </inline-definition>
```

```
    </ca-certs>
    <ee-certs>
      <inline-definition>
        <certificate>
          <name>Application #1</name>
          <cert-data>BASE64VALUE=</cert-data>
        </certificate>
        <certificate>
          <name>Application #2</name>
          <cert-data>BASE64VALUE=</cert-data>
        </certificate>
      </inline-definition>
    </ee-certs>
  </client-authentication>

  <keepalives>
    <max-wait>30</max-wait>
    <max-attempts>3</max-attempts>
  </keepalives>

</ssh-server>
```

The following configuration example uses central-keystore-references for the server identity and central-truststore-references for client authentication from the keystore:

```
================ NOTE: '\' line wrapping per RFC 8792 ================

<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<ssh-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">

  <!-- the host-key this SSH server will present -->
  <server-identity>
    <host-key>
      <name>my-pubkey-based-host-key</name>
      <public-key>
        <central-keystore-reference>ssh-rsa-key</central-keystore-re\
ference>
      </public-key>
    </host-key>
    <host-key>
      <name>my-cert-based-host-key</name>
      <certificate>
        <central-keystore-reference>
          <asymmetric-key>ssh-rsa-key-with-cert</asymmetric-key>
          <certificate>ex-rsa-cert2</certificate>
        </central-keystore-reference>
      </certificate>
    </host-key>
  </server-identity>

  <!-- the client credentials this SSH server will trust -->
  <client-authentication>
    <users>
```

```
      <user>
        <name>mary</name>
        <password>
          <hashed-password>$0$example-secret</hashed-password>
        </password>
        <public-keys>
          <central-truststore-reference>SSH Public Keys for Applicat\
ion A</central-truststore-reference>
        </public-keys>
      </user>
    </users>
    <ca-certs>
      <central-truststore-reference>trusted-client-ca-certs</central\
-truststore-reference>
    </ca-certs>
    <ee-certs>
      <central-truststore-reference>trusted-client-ee-certs</central\
-truststore-reference>
    </ee-certs>
  </client-authentication>

  <keepalives>
    <max-wait>30</max-wait>
    <max-attempts>3</max-attempts>
  </keepalives>

</ssh-server>
```

4.3. YANG Module

This YANG module has normative references to [RFC4251], [RFC4252], [RFC4253], [RFC4254], [RFC6991], [RFC7317], [RFC8341], [RFC9640], [RFC9641], and [RFC9642].

```
<CODE BEGINS> file "ietf-ssh-server@2024-03-16.yang"

module ietf-ssh-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
  prefix sshs;

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

  import iana-crypt-hash {
    prefix ianach;
    reference
      "RFC 7317: A YANG Data Model for System Management";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
```

```
      "RFC 8341: Network Configuration Access Control Model";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC 9640: YANG Data Types and Groupings for Cryptography";
  }

  import ietf-truststore {
    prefix ts;
    reference
      "RFC 9641: A YANG Data Model for a Truststore";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC 9642: A YANG Data Model for a Keystore";
  }

  import ietf-ssh-common {
    prefix sshcmn;
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

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

  contact
    "WG Web:   https://datatracker.ietf.org/wg/netconf
     WG List:  NETCONF WG list <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kent+ietf@watsen.net>";

  description
    "This module defines a reusable grouping for SSH servers that
     can be used as a basis for specific SSH server instances.

     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.

     Copyright (c) 2024 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 Revised
     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 9644
     (https://www.rfc-editor.org/info/rfc9644); see the RFC
```

```
      itself for full legal notices.";

  revision 2024-03-16 {
    description
      "Initial version.";
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }

  // Features

  feature ssh-server-keepalives {
    description
      "SSH keepalive parameters are configurable for
       SSH servers on the server implementing this feature.";
  }

  feature local-users-supported {
    description
      "Indicates that the configuration for users can be
       configured herein, as opposed to in an application-
       specific location.";
  }

  feature local-user-auth-publickey {
    if-feature "local-users-supported";
    description
      "Indicates that the 'publickey' authentication type,
       per RFC 4252, is supported for locally defined users.
       The 'publickey' authentication type is required by
       RFC 4252, but common implementations allow it to
       be disabled.";
    reference
      "RFC 4252:
        The Secure Shell (SSH) Authentication Protocol";
  }

  feature local-user-auth-password {
    if-feature "local-users-supported";
    description
      "Indicates that the 'password' authentication type,
       per RFC 4252, is supported for locally defined users.";
    reference
      "RFC 4252:
        The Secure Shell (SSH) Authentication Protocol";
  }

  feature local-user-auth-hostbased {
    if-feature "local-users-supported";
    description
      "Indicates that the 'hostbased' authentication type,
       per RFC 4252, is supported for locally defined users.";
    reference
      "RFC 4252:
        The Secure Shell (SSH) Authentication Protocol";
  }

  feature local-user-auth-none {
```

```
    if-feature "local-users-supported";
    description
      "Indicates that the 'none' authentication type, per
       RFC 4252, is supported.  It is NOT RECOMMENDED to
       enable this feature.";
    reference
      "RFC 4252:
        The Secure Shell (SSH) Authentication Protocol";
  }

  // Groupings

  grouping ssh-server-grouping {
    description
      "A reusable grouping for configuring an SSH server without
       any consideration for how underlying TCP sessions are
       established.

       Note that this grouping uses fairly typical descendant
       node names such that a nesting of 'uses' statements will
       have name conflicts.  It is intended that the consuming
       data model will resolve the issue (e.g., by wrapping
       the 'uses' statement in a container called
       'ssh-server-parameters').  This model purposely does
       not do this itself so as to provide maximum flexibility
       to consuming models.";

    container server-identity {
      nacm:default-deny-write;
      description
        "The list of host keys the SSH server will present when
         establishing an SSH connection.";
      list host-key {
        key "name";
        min-elements 1;
        ordered-by user;
        description
          "An ordered list of host keys (see RFC 4251) the SSH
           server will use to construct its ordered list of
           algorithms when sending its SSH_MSG_KEXINIT message,
           as defined in Section 7.1 of RFC 4253.";
        reference
          "RFC 4251: The Secure Shell (SSH) Protocol Architecture
           RFC 4253: The Secure Shell (SSH) Transport Layer
                     Protocol";
        leaf name {
          type string;
          description
            "An arbitrary name for this host key.";
        }
        choice host-key-type {
          mandatory true;
          description
            "The type of host key being specified.";
          container public-key {
            description
              "A locally defined or referenced asymmetric key pair
               to be used for the SSH server's host key.";
```

```
            reference
              "RFC 9642: A YANG Data Model for a Keystore";
            uses ks:inline-or-keystore-asymmetric-key-grouping {
              refine "inline-or-keystore/inline/inline-definition" {
                must 'not(public-key-format) or derived-from-or-self'
                 + '(public-key-format, "ct:ssh-public-key-format")';

              }
              refine "inline-or-keystore/central-keystore/"
                   + "central-keystore-reference" {
                must 'not(deref(.)/../ks:public-key-format) or '
                   + 'derived-from-or-self(deref(.)/../ks:public-'
                   + 'key-format, "ct:ssh-public-key-format")';
              }
            }
          }
          container certificate {
            if-feature "sshcmn:ssh-x509-certs";
            description
              "A locally defined or referenced end-entity
               certificate to be used for the SSH server's
               host key.";
            reference
              "RFC 9642: A YANG Data Model for a Keystore";
            uses
            ks:inline-or-keystore-end-entity-cert-with-key-grouping{
              refine "inline-or-keystore/inline/inline-definition" {
                must 'not(public-key-format) or derived-from-or-self'
                   + '(public-key-format, "ct:subject-public-key-'
                   + 'info-format")';
              }
              refine "inline-or-keystore/central-keystore/"
                   + "central-keystore-reference/asymmetric-key" {
                must 'not(deref(.)/../ks:public-key-format) or '
                   + 'derived-from-or-self(deref(.)/../ks:public-key'
                   + '-format, "ct:subject-public-key-info-format")';
              }
            }
          }
        }
      }
    } // container server-identity

    container client-authentication {
      nacm:default-deny-write;
      description
        "Specifies how the SSH server can be configured to
         authenticate SSH clients.  See RFC 4252 for a general
         discussion about SSH authentication.";
      reference
        "RFC 4252: The Secure Shell (SSH) Authentication Protocol";
      container users {
        if-feature "local-users-supported";
        description
          "A list of locally configured users.";
        list user {
          key "name";
          description
```

```
            "A locally configured user.

             The server SHOULD derive the list of authentication
             'method names' returned to the SSH client from the
             descendant nodes configured herein, per Sections
             5.1 and 5.2 of RFC 4252.

             The authentication methods are unordered.  Clients
             must authenticate to all configured methods.
             Whenever a choice amongst methods arises,
             implementations SHOULD use a default ordering
             that prioritizes automation over human interaction.";
          leaf name {
            type string;
            description
              "The 'username' for the SSH client, as defined in
               the SSH_MSG_USERAUTH_REQUEST message in RFC 4253.";
            reference
              "RFC 4253: The Secure Shell (SSH) Transport Layer
                         Protocol";
          }
          container public-keys {
            if-feature "local-user-auth-publickey";
            presence
              "Indicates that public keys have been configured.
               This statement is present so the mandatory descendant
               nodes do not imply that this node must be
               configured.";
            description
              "A set of SSH public keys may be used by the SSH
               server to authenticate this user.  A user is
               authenticated if its public key is an exact
               match to a configured public key.";
            reference
              "RFC 9641: A YANG Data Model for a Truststore";
            uses ts:inline-or-truststore-public-keys-grouping {
              refine "inline-or-truststore/inline/inline-definition/"
                   + "public-key" {
                must 'derived-from-or-self(public-key-format,'
                   + ' "ct:ssh-public-key-format")';
              }
              refine "inline-or-truststore/central-truststore/"
                   + "central-truststore-reference" {
                must 'not(deref(.)/../ts:public-key/ts:public-key-'
                   + 'format[not(derived-from-or-self(., "ct:ssh-'
                   + 'public-key-format"))])';
              }
            }
          }
          container password {
            description
              "A password the SSH server may use to authenticate
               this user.  A user is authenticated if the hash
               of the supplied password matches this value.";
            leaf hashed-password {
              if-feature "local-user-auth-password";
              type ianach:crypt-hash;
              description
```

```
                "The password for this user.";
            }
            leaf last-modified {
              type yang:date-and-time;
              config false;
              description
                "Identifies when the password was last set.";
            }
          }
          container hostbased {
            if-feature "local-user-auth-hostbased";
            presence
              "Indicates that host-based (RFC 4252) keys have been
               configured.  This statement is present so the
               mandatory descendant nodes do not imply that this
               node must be configured.";
            description
              "A set of SSH host keys used by the SSH server to
               authenticate this user's host.  A user's host is
               authenticated if its host key is an exact match
               to a configured host key.";
            reference
              "RFC 4252: The Secure Shell (SSH) Authentication
                         Protocol
               RFC 9641: A YANG Data Model for a Truststore";
            uses ts:inline-or-truststore-public-keys-grouping {
              refine "inline-or-truststore/inline/inline-definition/"
                   + "public-key" {
                must 'derived-from-or-self(public-key-format,'
                   + ' "ct:ssh-public-key-format")';
              }
              refine "inline-or-truststore/central-truststore/"
                   + "central-truststore-reference" {
                must 'not(deref(.)/../ts:public-key/ts:public-key-'
                   + 'format[not(derived-from-or-self(., "ct:ssh-'
                   + 'public-key-format"))])';
              }
            }
          }
          leaf none {
            if-feature "local-user-auth-none";
            type empty;
            description
              "Indicates that the 'none' method is configured
               for this user.";
            reference
              "RFC 4252: The Secure Shell (SSH) Authentication
                         Protocol";
          }
        }
      } // users
      container ca-certs {
        if-feature "sshcmn:ssh-x509-certs";
        presence
          "Indicates that CA certificates have been configured.
           This statement is present so the mandatory descendant
           nodes do not imply this node must be configured.";
        description
```

```
          "A set of Certification Authority (CA) certificates used by
           the SSH server to authenticate SSH client certificates.
           A client certificate is authenticated if it has a valid
           chain of trust to a configured CA certificate.";
        reference
          "RFC 9641: A YANG Data Model for a Truststore";
        uses ts:inline-or-truststore-certs-grouping;
      }
      container ee-certs {
        if-feature "sshcmn:ssh-x509-certs";
        presence
          "Indicates that EE certificates have been configured.
           This statement is present so the mandatory descendant
           nodes do not imply this node must be configured.";
        description
          "A set of client certificates (i.e., end-entity
           certificates) used by the SSH server to authenticate
           the certificates presented by SSH clients.  A client
           certificate is authenticated if it is an exact match
           to a configured end-entity certificate.";
        reference
          "RFC 9641: A YANG Data Model for a Truststore";
        uses ts:inline-or-truststore-certs-grouping;
      }
    } // container client-authentication

    container transport-params {
      nacm:default-deny-write;
      if-feature "sshcmn:transport-params";
      description
        "Configurable parameters of the SSH transport layer.";
      uses sshcmn:transport-params-grouping;
    } // container transport-params

    container keepalives {
      nacm:default-deny-write;
      if-feature "ssh-server-keepalives";
      presence
        "Indicates that the SSH server proactively tests the
         aliveness of the remote SSH client.";
      description
        "Configures the keepalive policy to proactively test
         the aliveness of the SSH client.  An unresponsive SSH
         client is dropped after approximately max-wait *
         max-attempts seconds.  Per Section 4 of RFC 4254,
         the SSH server SHOULD send an SSH_MSG_GLOBAL_REQUEST
         message with a purposely nonexistent 'request name'
         value (e.g., keepalive@example.com) and the 'want reply'
         value set to '1'.";
      reference
        "RFC 4254: The Secure Shell (SSH) Connection Protocol";
      leaf max-wait {
        type uint16 {
          range "1..max";
        }
        units "seconds";
        default "30";
        description
```

```
            "Sets the amount of time in seconds after which
             an SSH-level message will be sent to test the
             aliveness of the SSH client if no data has been
             received from the SSH client.";
        }
        leaf max-attempts {
          type uint8;
          default "3";
          description
            "Sets the maximum number of sequential keepalive
             messages that can fail to obtain a response from
             the SSH client before assuming the SSH client is
             no longer alive.";
        }
      }
    } // grouping ssh-server-grouping

}

<CODE ENDS>
```

5. Security Considerations

The three IETF YANG modules in this document define groupings and will not be deployed as standalone modules. Their security implications may be context-dependent based on their use in other modules. The designers of modules that import these groupings must conduct their own analysis of the security considerations.

5.1. Considerations for the "iana-ssh-key-exchange-algs" Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "iana-ssh-key-exchange-algs" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

This YANG module defines YANG enumerations for a public IANA-maintained registry.

YANG enumerations are not security-sensitive, as they are statically defined in the publicly accessible YANG module. IANA **MAY** deprecate and/or obsolete enumerations over time as needed to address security issues found in the algorithms.

This module does not define any writable nodes, RPCs, actions, or notifications, and thus, the security considerations for such are not provided here.

5.2. Considerations for the "iana-ssh-encryption-algs" Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "iana-ssh-encryption-algs" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

This YANG module defines YANG enumerations for a public IANA-maintained registry.

YANG enumerations are not security-sensitive, as they are statically defined in the publicly accessible YANG module.

This module does not define any writable nodes, RPCs, actions, or notifications, and thus, the security considerations for such are not provided here.

5.3. Considerations for the "iana-ssh-mac-algs" Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "iana-ssh-mac-algs" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

This YANG module defines YANG enumerations for a public IANA-maintained registry.

YANG enumerations are not security-sensitive, as they are statically defined in the publicly accessible YANG module.

This module does not define any writable nodes, RPCs, actions, or notifications, and thus, the security considerations for such are not provided here.

5.4. Considerations for the "iana-ssh-public-key-algs" Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "iana-ssh-public-key-algs" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

This YANG module defines YANG enumerations for a public IANA-maintained registry.

YANG enumerations are not security-sensitive, as they are statically defined in the publicly accessible YANG module.

This module does not define any writable nodes, RPCs, actions, or notifications, and thus, the security considerations for such are not provided here.

5.5. Considerations for the "ietf-ssh-common" YANG Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "ietf-ssh-common" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

Please be aware that this YANG module uses groupings from other YANG modules that define nodes that may be considered sensitive or vulnerable in network environments. Please review the security considerations for dependent YANG modules for information as to which nodes may be considered sensitive or vulnerable in network environments.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

None of the writable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-write" extension has not been set for any data nodes defined in this module.

This module defines the "generate-asymmetric-key-pair" RPC, which may, if the "ct:cleartext-private-keys" feature is enabled and the client requests it, return the private clear in cleartext form. It is **NOT RECOMMENDED** for private keys to pass the server's security perimeter.

This module does not define any actions or notifications, and thus, the security considerations for such are not provided here.

5.6. Considerations for the "ietf-ssh-client" YANG Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "ietf-ssh-client" YANG module defines "grouping" statements that are designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

Please be aware that this YANG module uses groupings from other YANG modules that define nodes that may be considered sensitive or vulnerable in network environments. Please review the security considerations for dependent YANG modules for information as to which nodes may be considered sensitive or vulnerable in network environments.

One readable data node defined in this YANG module may be considered sensitive or vulnerable in some network environments. This node is as follows:

- The "client-identity/password" node:

  The cleartext "password" node defined in the "ssh-client-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.

All the writable data nodes defined by this module may be considered sensitive or vulnerable in some network environments. For instance, any modification to a key or reference to a key may dramatically alter the implemented security policy. For this reason, the NACM extension "default-deny-write" has been set for all data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus, the security considerations for such are not provided here.

5.7. Considerations for the "ietf-ssh-server" YANG Module

This section is modeled after the template defined in Section 3.7.1 of [RFC8407].

The "ietf-ssh-server" YANG module defines "grouping" statements that are designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have mandatory-to-implement secure transport layers (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and mandatory-to-implement mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

Please be aware that this YANG module uses groupings from other YANG modules that define nodes that may be considered sensitive or vulnerable in network environments. Please review the security considerations for dependent YANG modules for information as to which nodes may be considered sensitive or vulnerable in network environments.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

All the writable data nodes defined by this module may be considered sensitive or vulnerable in some network environments. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., or even the modification of transport or keepalive parameters can dramatically alter the implemented security policy. For this reason, the NACM extension "default-deny-write" has been set for all data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus, the security considerations for such are not provided here.

6. IANA Considerations

6.1. The IETF XML Registry

IANA has registered seven URIs in the "ns" registry of the "IETF XML Registry" [RFC3688] as follows.

URI: urn:ietf:params:xml:ns:yang:iana-ssh-key-exchange-algs
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:iana-ssh-encryption-algs
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:iana-ssh-mac-algs
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:iana-ssh-public-key-algs
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-common
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-client
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-server
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

6.2. The YANG Module Names Registry

IANA has registered seven YANG modules in the "YANG Module Names" registry [RFC6020] as follows.

Name: iana-ssh-key-exchange-algs
Namespace: urn:ietf:params:xml:ns:yang:iana-ssh-key-exchange-algs
Prefix: sshkea
Reference: RFC 9644

Name: iana-ssh-encryption-algs
Namespace: urn:ietf:params:xml:ns:yang:iana-ssh-encryption-algs
Prefix: sshea
Reference: RFC 9644

Name: iana-ssh-mac-algs
Namespace: urn:ietf:params:xml:ns:yang:iana-ssh-mac-algs
Prefix: sshma
Reference: RFC 9644

Name: iana-ssh-public-key-algs
Namespace: urn:ietf:params:xml:ns:yang:iana-ssh-public-key-algs
Prefix: sshpka
Reference: RFC 9644

Name: ietf-ssh-common
Namespace: urn:ietf:params:xml:ns:yang:ietf-ssh-common
Prefix: sshcmn
Reference: RFC 9644

Name: ietf-ssh-client
Namespace: urn:ietf:params:xml:ns:yang:ietf-ssh-client
Prefix: sshc
Reference: RFC 9644

Name: ietf-ssh-server
Namespace: urn:ietf:params:xml:ns:yang:ietf-ssh-server
Prefix: sshs
Reference: RFC 9644

6.3. Considerations for the "iana-ssh-encryption-algs" Module

This section follows the template defined in Section 4.30.3.1 of [YANG-GUIDE].

This document presents a script (see Appendix A) for IANA to use to generate the IANA-maintained "iana-ssh-encryption-algs" YANG module. The most recent version of the YANG module is available in the "YANG Parameters" registry group [IANA-YANG-PARAMETERS].

IANA has added the following note to the registry:

> New values must not be directly added to the "iana-ssh-encryption-algs" YANG module. They must instead be added to the "Encryption Algorithm Names" registry of the "Secure Shell (SSH) Protocol Parameters" registry group [IANA-ENC-ALGS].

When a value is added to the "Encryption Algorithm Names" registry, a new "enum" statement must be added to the "iana-ssh-encryption-algs" YANG module. The "enum" statement, and substatements thereof, should be defined as follows:

enum
: Replicates a name from the registry.

value
: Contains the decimal value of the IANA-assigned value.

status
: Include only if a registration has been deprecated or obsoleted. An IANA "Note" containing the word "HISTORIC" maps to YANG status "obsolete". Since the registry is unable to express a "**SHOULD NOT**" recommendation, there is no mapping to YANG status "deprecated".

description
: Contains "Enumeration for the 'foo-bar' algorithm.", where "foo-bar" is a placeholder for the algorithm's name (e.g., "3des-cbc").

reference
: Replicates the reference(s) from the registry with the title of the document(s) added.

Unassigned or reserved values are not present in the module.

When the "iana-ssh-encryption-algs" YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements. The "revision" must have a "description" statement explaining why the update occurred and must have a "reference" substatement that points to the document defining the registry update that resulted in this change. For instance:

```
revision 2024-02-02 {
    description
      "This update reflects the update made to the underlying
       Foo Bar registry per RFC XXXX.";
  reference
      "RFC XXXX: Extend the Foo Bars Registry
                 to Support Something Important";
}
```

IANA has added the following note to the "Encryption Algorithm Names" registry.

> When this registry is modified, the YANG module "iana-ssh-encryption-algs" [IANA-YANG-PARAMETERS] must be updated as defined in RFC 9644.

6.4. Considerations for the "iana-ssh-mac-algs" Module

This section follows the template defined in Section 4.30.3.1 of [YANG-GUIDE].

This document presents a script (see Appendix A) for IANA to use to generate the IANA-maintained "iana-ssh-mac-algs" YANG module. The most recent version of the YANG module is available in the "YANG Parameters" registry group [IANA-YANG-PARAMETERS].

IANA has added the following note to the registry:

> New values must not be directly added to the "iana-ssh-mac-algs" YANG module. They must instead be added to the "MAC Algorithm Names" registry of the "Secure Shell (SSH) Protocol Parameters" registry group [IANA-MAC-ALGS].

When a value is added to the "MAC Algorithm Names" registry, a new "enum" statement must be added to the "iana-ssh-mac-algs" YANG module. The "enum" statement, and substatements thereof, should be defined as follows:

enum
: Replicates a name from the registry.

value
: Contains the decimal value of the IANA-assigned value.

status
: Include only if a registration has been deprecated or obsoleted.

description
: Contains "Enumeration for the 'foo-bar' algorithm.", where "foo-bar" is a placeholder for the algorithm's name (e.g., "3des-cbc").

reference
: Replicates the reference(s) from the registry with the title of the document(s) added.

Unassigned or reserved values are not present in the module.

When the "iana-ssh-mac-algs" YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements. The "revision" must have a "description" statement explaining why the update occurred and must have a "reference" substatement that points to the document defining the registry update that resulted in this change. For instance:

```
revision 2024-02-02 {
    description
      "This update reflects the update made to the underlying
       Foo Bar registry per RFC XXXX.";
  reference
      "RFC XXXX: Extend the Foo Bars Registry
                 to Support Something Important";
}
```

IANA has added the following note to the "MAC Algorithm Names" registry.

> When this registry is modified, the YANG module "iana-ssh-mac-algs" [IANA-YANG-PARAMETERS] must be updated as defined in RFC 9644.

6.5. Considerations for the "iana-ssh-public-key-algs" Module

This section follows the template defined in Section 4.30.3.1 of [YANG-GUIDE].

This document presents a script (see Appendix A) for IANA to use to generate the IANA-maintained "iana-ssh-public-key-algs" YANG module. The most recent version of the YANG module is available in the "YANG Parameters" registry group [IANA-YANG-PARAMETERS].

IANA has added the following note to the registry:

> New values must not be directly added to the "iana-ssh-public-key-algs" YANG module. They must instead be added to the "Public Key Algorithm Names" registry of the "Secure Shell (SSH) Protocol Parameters" registry group [IANA-PUBKEY-ALGS].

When a value is added to the "Public Key Algorithm Names" registry, a new "enum" statement must be added to the "iana-ssh-public-key-algs" YANG module. The "enum" statement, and substatements thereof, should be defined as follows:

enum
: Replicates a name from the registry.

value
: Contains the decimal value of the IANA-assigned value.

status
: Include only if a registration has been deprecated or obsoleted.

description
: Contains "Enumeration for the 'foo-bar' algorithm.", where "foo-bar" is a placeholder for the algorithm's name (e.g., "3des-cbc").

reference
: Replicates the reference(s) from the registry with the title of the document(s) added.

In the case that the algorithm name ends with "-*", the family of enumerations must be added. The family of enum algorithm names are generated by replacing the "*" character with these strings: "nistp256", "nistp384", "nistp521", "1.3.132.0.1", "1.2.840.10045.3.1.1", "1.3.132.0.33", "1.3.132.0.26", "1.3.132.0.27", "1.3.132.0.16", "1.3.132.0.36", "1.3.132.0.37", and "1.3.132.0.38".

Unassigned or reserved values are not present in the module.

When the "iana-ssh-public-key-algs" YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements. The "revision" must have a "description" statement explaining why the update occurred and must have a "reference" substatement that points to the document defining the registry update that resulted in this change. For instance:

```
revision 2024-02-02 {
    description
      "This update reflects the update made to the underlying
       Foo Bar registry per RFC XXXX.";
  reference
      "RFC XXXX: Extend the Foo Bars Registry
                 to Support Something Important";
}
```

IANA has added the following note to the "Public Key Algorithm Names" registry.

> When this registry is modified, the YANG module "iana-ssh-public-key-algs" [IANA-YANG-PARAMETERS] must be updated as defined in RFC 9644.

6.6. Considerations for the "iana-ssh-key-exchange-algs" Module

This section follows the template defined in Section 4.30.3.1 of [YANG-GUIDE].

This document presents a script (see Appendix A) for IANA to use to generate the IANA-maintained "iana-ssh-key-exchange-algs" YANG module. The most recent version of the YANG module is available in the "YANG Parameters" registry group [IANA-YANG-PARAMETERS].

IANA has added the following note to the registry:

> New values must not be directly added to the "iana-ssh-key-exchange-algs" YANG module. They must instead be added to the "Key Exchange Method Names" registry of the "Secure Shell (SSH) Protocol Parameters" registry group [IANA-KEYEX-ALGS].

When a value is added to the "Key Exchange Method Names" registry, a new "enum" statement must be added to the "iana-ssh-key-exchange-algs" YANG module. The "enum" statement, and substatements thereof, should be defined as follows:

enum
: Replicates a name from the registry.

value
: Contains the decimal value of the IANA-assigned value.

status
: Include only if a registration has been deprecated or obsoleted. An IANA "OK to Implement" containing "**SHOULD NOT**" maps to YANG status "deprecated". An IANA "OK to Implement" containing "**MUST NOT**" maps to YANG status "obsolete".

description
: Contains "Enumeration for the 'foo-bar' algorithm.", where "foo-bar" is a placeholder for the algorithm's name (e.g., "3des-cbc").

reference
: Replicates the reference(s) from the registry with the title of the document(s) added.

In the case that the algorithm name ends with "-*", the family of enumerations must be added. The family of enum algorithm names are generated by replacing the "*" character with these strings: "nistp256", "nistp384", "nistp521", "1.3.132.0.1", "1.2.840.10045.3.1.1", "1.3.132.0.33", "1.3.132.0.26", "1.3.132.0.27", "1.3.132.0.16", "1.3.132.0.36", "1.3.132.0.37", and "1.3.132.0.38".

Unassigned or reserved values are not present in the module.

When the "iana-ssh-key-exchange-algs" YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements. The "revision" must have a "description" statement explaining why the update occurred, and must have a "reference" substatement that points to the document defining the registry update that resulted in this change. For instance:

```
revision 2024-02-02 {
    description
      "This update reflects the update made to the underlying
       Foo Bar registry per RFC XXXX.";
  reference
      "RFC XXXX: Extend the Foo Bars Registry
                 to Support Something Important";
}
```

IANA has added the following note to the "Key Exchange Method Names" registry.

> When this registry is modified, the YANG module "iana-ssh-key-exchange-algs" [IANA-YANG-PARAMETERS] must be updated as defined in RFC 9644.

7. References

7.1. Normative References

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

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, DOI 10.17487/RFC4250, January 2006, <https://www.rfc-editor.org/info/rfc4250>.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <https://www.rfc-editor.org/info/rfc4251>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[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>.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <https://www.rfc-editor.org/info/rfc4254>.

[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>.

[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>.

[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>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

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

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[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>.

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

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

[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>.

[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>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9640] Watsen, K., "YANG Data Types and Groupings for Cryptography", RFC 9640, DOI 10.17487/RFC9640, September 2024, <https://www.rfc-editor.org/info/rfc9640>.

[RFC9641] Watsen, K., "A YANG Data Model for a Truststore", RFC 9641, DOI 10.17487/RFC9641, September 2024, <https://www.rfc-editor.org/info/rfc9641>.

[RFC9642] Watsen, K., "A YANG Data Model for a Keystore", RFC 9642, DOI 10.17487/RFC9642, September 2024, <https://www.rfc-editor.org/info/rfc9642>.

7.2. Informative References

[FIPS_186-5] NIST, "Digital Signature Standard (DSS)", FIPS PUB 186-5, DOI 10.6028/NIST.FIPS.186-5, February 2023, <https://csrc.nist.gov/pubs/fips/186-5/final>.

[HTTP-CLIENT-SERVER] Watsen, K., "YANG Groupings for HTTP Clients and HTTP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-http-client-server-23, 15 August 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-http-client-server-23>.

[IANA-ENC-ALGS] IANA, "Encryption Algorithm Names", <https://www.iana.org/assignments/ssh-parameters/>.

[IANA-KEYEX-ALGS] IANA, "Key Exchange Method Names", <https://www.iana.org/assignments/ssh-parameters>.

[IANA-MAC-ALGS] IANA, "MAC Algorithm Names", <https://www.iana.org/assignments/ssh-parameters>.

[IANA-PUBKEY-ALGS] IANA, "Public Key Algorithm Names", <https://www.iana.org/assignments/ssh-parameters/>.

[IANA-YANG-PARAMETERS] IANA, "YANG Parameters", <https://www.iana.org/assignments/yang-parameters>.

[NETCONF-CLIENT-SERVER] Watsen, K., "NETCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-netconf-client-server-37, 14 August 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-netconf-client-server-37>.

[RESTCONF-CLIENT-SERVER] Watsen, K., "RESTCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-client-server-38, 14 August 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-restconf-client-server-38>.

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <https://www.rfc-editor.org/info/rfc8071>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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

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

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[RFC9643] Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", RFC 9643, DOI 10.17487/RFC9643, September 2024, <https://www.rfc-editor.org/info/rfc9643>.

[RFC9645] Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", RFC 9645, DOI 10.17487/RFC9645, September 2024, <https://www.rfc-editor.org/info/rfc9645>.

[SYSTEM-CONFIG] Ma, Q., Wu, Q., and C. Feng, "System-defined Configuration", Work in Progress, Internet-Draft, draft-ietf-netmod-system-config-08, 18 June 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-system-config-08>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

[YANG-GUIDE] Bierman, A., Boucadair, M., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc8407bis-15, 10 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc8407bis-15>.

Appendix A. Script to Generate IANA-Maintained YANG Modules

This section is not normative.

The Python <https://www.python.org> script contained in this section will create the four IANA-maintained modules that are described (but not contained) in this document.

Run the script using the command "python gen-yang-modules.py" to produce four YANG module files in the current directory.

Be aware that the script does not attempt to copy the "revision" statements from the previous/current YANG module. Copying the revision statements must be done manually.

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

import re
import csv
import textwrap
import requests
import requests_cache
from io import StringIO
from datetime import datetime

# Metadata for the four YANG modules produced by this script
MODULES = [
    {
        "csv_url": "https://www.iana.org/assignments/ssh-parameters/\
ssh-parameters-17.csv",
        "spaced_name": "encryption",
        "hypenated_name": "encryption",
        "prefix": "sshea",
        "description": """    "This module defines enumerations for \
the encryption algorithms
     defined in the 'Encryption Algorithm Names' registry of the
     'Secure Shell (SSH) Protocol Parameters' registry group
     maintained by IANA.""",
    },
    {
        "csv_url": "https://www.iana.org/assignments/ssh-parameters/\
ssh-parameters-19.csv",
        "spaced_name": "public key",
        "hypenated_name": "public-key",
        "prefix": "sshpka",
        "description": """    "This module defines enumerations for \
the public key algorithms
     defined in the 'Public Key Algorithm Names' registry of the
     'Secure Shell (SSH) Protocol Parameters' registry group
     maintained by IANA."""
    },
    {
        "csv_url": "https://www.iana.org/assignments/ssh-parameters/\
ssh-parameters-18.csv",
        "spaced_name": "mac",
        "hypenated_name": "mac",
        "prefix": "sshma",
        "description": """    "This module defines enumerations for \
the MAC algorithms
     defined in the 'MAC Algorithm Names' registry of the
     'Secure Shell (SSH) Protocol Parameters' registry group
     maintained by IANA."""
    },
    {
        "csv_url": "https://www.iana.org/assignments/ssh-parameters/\
ssh-parameters-16.csv",
        "spaced_name": "key exchange",
        "hypenated_name": "key-exchange",
        "prefix": "sshkea",
        "description": """    "This module defines enumerations for \
the key exchange algorithms
```

```
     defined in the 'Key Exchange Method Names' registry of the
     'Secure Shell (SSH) Protocol Parameters' registry group
     maintained by IANA."""
    },
]



def create_module_begin(module, f):

    # Define template for all four modules
    PREAMBLE_TEMPLATE="""
module iana-ssh-HNAME-algs {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:iana-ssh-HNAME-algs";
  prefix PREFIX;

  organization
    "Internet Assigned Numbers Authority (IANA)";

  contact
    "Postal: ICANN
             12025 Waterfront Drive, Suite 300
             Los Angeles, CA  90094-2536
             United States of America
     Tel:    +1 310 301 5800
     Email:  iana@iana.org";

  description
DESCRIPTION

     Copyright (c) YEAR 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 Revised
     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).

     The initial version of this YANG module is part of RFC 9644
     (https://www.rfc-editor.org/info/rfc9644); see the RFC
     itself for full legal notices.

     All versions of this module are published by IANA at
     https://www.iana.org/assignments/yang-parameters.";

  revision DATE {
    description
      "This initial version of the module was created using
       the script defined in RFC 9644 to reflect the contents
       of the SNAME algorithms registry maintained by IANA.";
    reference
      "RFC 9644: YANG Groupings for SSH Clients and SSH Servers";
  }
```

```
  typedef ssh-HNAME-algorithm {
    type enumeration {
"""
    # Replacements
    rep = {
      "DATE": datetime.today().strftime('%Y-%m-%d'),
      "YEAR": datetime.today().strftime('%Y'),
      "SNAME": module["spaced_name"],
      "HNAME": module["hypenated_name"],
      "PREFIX": module["prefix"],
      "DESCRIPTION": module["description"]
    }

    # Do the replacement
    rep = dict((re.escape(k), v) for k, v in rep.items())
    pattern = re.compile("|".join(rep.keys()))
    text = pattern.sub(lambda m: rep[re.escape(m.group(0))], PREAMBL\
E_TEMPLATE)

    # Write preamble into the file
    f.write(text)


def create_module_body(module, f):

    # Fetch the current CSV file from IANA
    r = requests.get(module["csv_url"])
    assert(r.status_code == 200)

    # Ascertain the first CSV column's name
    with StringIO(r.text) as csv_file:
        csv_reader = csv.reader(csv_file)
        for row in csv_reader:
            first_colname = row[0]
            break

    # Parse each CSV line
    with StringIO(r.text) as csv_file:
        csv_reader = csv.DictReader(csv_file)
        for row in csv_reader:

            # Extract just the ref
            refs = row["Reference"][1:-1]  # remove the '[' and ']' \
chars
            refs = refs.split("][")

            # There may be more than one ref
            titles = []
            for ref in refs:

                # Ascertain the ref's title
                if ref.startswith("RFC"):

                    # Fetch the current BIBTEX entry
                    bibtex_url="https://datatracker.ietf.org/doc/"+ \
ref.lower() + "/bibtex/"
                    r = requests.get(bibtex_url)
                    assert r.status_code == 200, "Could not GET " + \
```

```
bibtex_url

                    # Append to 'titles' value from the "title" line
                    for item in r.text.split("\n"):
                        if "title =" in item:
                            titles.append(re.sub('.*{{(.*)}}.*', r'\\
g<1>', item))
                            break
                    else:
                        raise Exception("RFC title not found")


                    # Insert a space: "RFCXXXX" --> "RFC XXXX"
                    index = refs.index(ref)
                    refs[index] = "RFC " + ref[3:]

                elif ref.startswith("FIPS"):
                    # Special case for FIPS, since no bibtex to fetch
                    if ref == "FIPS 46-3" or ref == "FIPS-46-3":
                        titles.append("Data Encryption Standard (DES\
)")
                    else:
                        raise Exception("FIPS ref not found")

                else:
                    raise Exception("ref not found")

            # Function used below
            def write_enumeration(alg):
                f.write('\n')
                f.write(f'      enum {alg} {{\n')
                if "HISTORIC" in row["Note"]:
                    f.write(f'        status obsolete;\n')
                elif "OK to Implement" in row:
                    if "MUST NOT" in row["OK to Implement"]:
                        f.write(f'        status obsolete;\n')
                    elif "SHOULD NOT" in row["OK to Implement"]:
                        f.write(f'        status deprecated;\n')
                f.write(f'        description\n')
                description = f'          "Enumeration for the \'{al\
g}\' algorithm.'
                if "Section" in row["Note"]:
                    description += " " + row["Note"]
                description += '";'
                description = textwrap.fill(description, width=69, s\
ubsequent_indent="           ")
                f.write(f'{description}\n')
                f.write('        reference\n')
                f.write('          "')
                if row["Reference"] == "":
                    f.write('    Missing in IANA registry.')
                else:
                    ref_len = len(refs)
                    for i in range(ref_len):
                        ref = refs[i]
                        f.write(f'{ref}:\n')
                        title = "            " + titles[i]
                        if i == ref_len - 1:
```

```
                            title += '";'
                        title = textwrap.fill(title, width=67, subse\
quent_indent="                  ")
                        f.write(f'{title}')
                        if i != ref_len - 1:
                            f.write('\n       ')
                f.write('\n')
                f.write('      }\n')


            # Write one or more "enumeration" statements
            if not row[first_colname].endswith("-*"): # just one enu\
meration
                # Avoid duplicate entries caused by the "ecdh-sha2-*\
" family expansion
                if not row[first_colname].startswith("ecdh-sha2-nist\
p"):
                    write_enumeration(row[first_colname])
            else: # a family of enumerations
                curve_ids = [
                    "nistp256",
                    "nistp384",
                    "nistp521",
                    "1.3.132.0.1",
                    "1.2.840.10045.3.1.1",
                    "1.3.132.0.33",
                    "1.3.132.0.26",
                    "1.3.132.0.27",
                    "1.3.132.0.16",
                    "1.3.132.0.36",
                    "1.3.132.0.37",
                    "1.3.132.0.38",
                ]
                for curve_id in curve_ids:
                    write_enumeration(row[first_colname][:-1] + curv\
e_id)


def create_module_end(module, f):

    # Close out the enumeration, typedef, and module
    f.write("    }\n")
    f.write("    description\n")
    f.write(f'      "An enumeration for SSH {module["spaced_name"]} \
algorithms.";\n')
    f.write("  }\n")
    f.write('\n')
    f.write('}\n')


def create_module(module):

    # Install cache for 8x speedup
    requests_cache.install_cache()

    # Ascertain YANG module's name
    yang_module_name = "iana-ssh-" + module["hypenated_name"] + "-al\
gs.yang"
```

```
    # Create YANG module file
    with open(yang_module_name, "w") as f:
        create_module_begin(module, f)
        create_module_body(module, f)
        create_module_end(module, f)


def main():
    for module in MODULES:
        create_module(module)


if __name__ == "__main__":
    main()

<CODE ENDS>
```

Acknowledgements

The authors would like to thank the following for lively discussions on list and in the halls (ordered by first name): Alan Luchuk, Andy Bierman, Balázs Kovács, Barry Leiba, Benoit Claise, Bert Wijnen, David Lamparter, Elwyn Davies, Gary Wu, Jürgen Schönwälder, Ladislav Lhotka, Liang Xia, Martin Björklund, Martin Thomson, Mehmet Ersue, Michal Vaško, Murray Kucherawy, Paul Wouters, Per Andersson, Phil Shafer, Qin Wun, Radek Krejci, Rob Wilton, Roman Danyliw, Russ Housley, Sean Turner, Thomas Martin, Tom Petch, and Warren Kumari.

Contributors

Special acknowledgement goes to Gary Wu for his work on the "ietf-ssh-common" module.

Author's Address

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net