Workgroup: NETCONF Working Group
Internet-Draft:
draft-ietf-netconf-ssh-client-server-24
Published: 18 May 2021
Intended Status: Standards Track
Expires: 19 November 2021
A K. Watsen
uWatsen Networks
t
h
o
r
s
:

YANG Groupings for SSH Clients and SSH Servers

Abstract

This document defines three YANG 1.1 modules: the first defines identities and groupings common to both SSH clients and SSH servers, the second defines a grouping for a generic SSH client, and the third defines a grouping for a generic SSH server.

Editorial Note (To be removed by RFC Editor)

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

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

* AAAA --> the assigned RFC value for draft-ietf-netconf-crypto-types
* BBBB --> the assigned RFC value for draft-ietf-netconf-trust-anchors
* CCCC --> the assigned RFC value for draft-ietf-netconf-keystore
* DDDD --> the assigned RFC value for draft-ietf-netconf-tcp-client-server
* EEEE --> the assigned RFC value for this draft

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

* 2021-05-18 --> the publication date of this draft

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

* Appendix A. Change Log

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines three YANG 1.1 [RFC7950] modules: the first defines identities and groupings common to both SSH clients and SSH servers, the second defines a grouping for a generic SSH client, and the third defines a grouping for a generic SSH server. It is intended that these groupings will be used by applications using the SSH protocol [RFC4252], [RFC4253], and [RFC4254]. For instance, these groupings could be used to help define the data model for an OpenSSH [OPENSSH] server or a NETCONF over SSH [RFC6242] based server.

The client and server YANG modules in this document each define one grouping, which is focused on just SSH-specific configuration, and specifically avoids 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 use groupings defined in [I-D.ietf-netconf-keystore] enabling keys to be either locally defined or a reference to globally configured values.

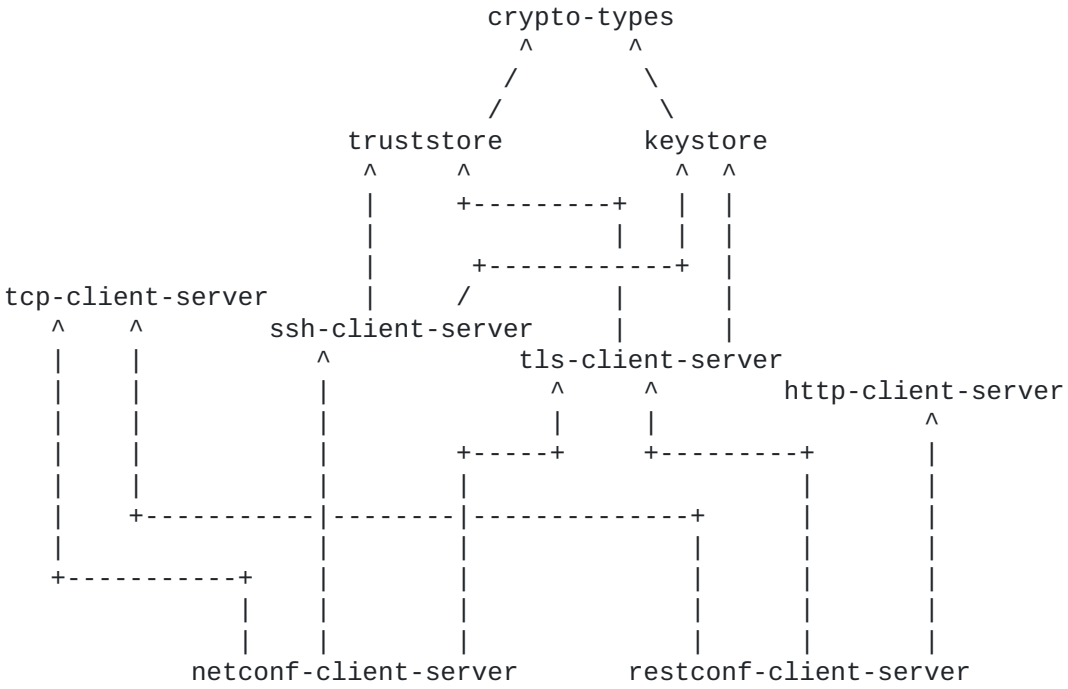
The modules defined in this document optionally support [RFC6187] enabling X.509v3 certificate based host keys and public keys.

1.1. Relation to other RFCs

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

The modules have been defined in a modular fashion to enable their use by other efforts, some of which are known to be in progress at the time of this writing, with many more expected to be defined in time.

The normative dependency relationship between the various RFCs in the collection is presented in the below diagram. The labels in the diagram represent the primary purpose provided by each RFC. Hyperlinks to each RFC are provided below the diagram.



Label in Diagram	Originating RFC
crypto-types	[I-D.ietf-netconf-crypto-types]
truststore	[I-D.ietf-netconf-trust-anchors]
keystore	[I-D.ietf-netconf-keystore]
tcp-client-server	[I-D.ietf-netconf-tcp-client-server]
ssh-client-server	[I-D.ietf-netconf-ssh-client-server]
tls-client-server	[I-D.ietf-netconf-tls-client-server]
http-client-server	[I-D.ietf-netconf-http-client-server]
netconf-client-server	[I-D.ietf-netconf-netconf-client-server]
restconf-client-server	[I-D.ietf-netconf-restconf-client-server]

Table 1: Label to RFC Mapping

1.2. 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.

1.3. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, as described in [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore], trust anchors and keys installed during manufacturing are expected to appear in <operational>.

2. The "ietf-ssh-common" Module

The SSH common model presented in this section contains identities and groupings 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 algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server 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.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive. As well, some algorithms that are REQUIRED by [RFC4253] are missing, notably "ssh-dss" and "diffie-hellman-group1-sha1" due to their weak security and there being alternatives that are widely supported.

2.1. Data Model Overview

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

2.1.1. Features

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

```
Features:
  +-- ssh-ecc
  +-- ssh-x509-certs
  +-- ssh-dh-group-exchange
  +-- ssh-ctr
  +-- ssh-sha2
```

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

2.1.2. Identities

The following diagram illustrates the relationship amongst the "identity" statements defined in the "ietf-ssh-common" module:

```
Identities:
  +-- public-key-alg-base
  |  +-- ssh-dss
  |  +-- ssh-rsa
  |  +-- ecdsa-sha2-nistp256
  |  +-- ecdsa-sha2-nistp384
  |  +-- ecdsa-sha2-nistp521
  |  +-- x509v3-ssh-rsa
  |  +-- x509v3-rsa2048-sha256
  |  +-- x509v3-ecdsa-sha2-nistp256
  |  +-- x509v3-ecdsa-sha2-nistp384
  |  +-- x509v3-ecdsa-sha2-nistp521
  +-- key-exchange-alg-base
  |  +-- diffie-hellman-group14-sha1
  |  +-- diffie-hellman-group-exchange-sha1
  |  +-- diffie-hellman-group-exchange-sha256
  |  +-- ecdh-sha2-nistp256
  |  +-- ecdh-sha2-nistp384
  |  +-- ecdh-sha2-nistp521
  +-- encryption-alg-base
  |  +-- triple-des-cbc
  |  +-- aes128-cbc
  |  +-- aes192-cbc
  |  +-- aes256-cbc
  |  +-- aes128-ctr
  |  +-- aes192-ctr
  |  +-- aes256-ctr
  +-- mac-alg-base
     +-- hmac-sha1
     +-- hmac-sha2-256
     +-- hmac-sha2-512
```

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

Comments:

- The diagram shows that there are four base identities.
- These identities are used by this module to define algorithms for public-key, key-exchange, encryption, and MACs.
- These base identities are "abstract", in the object oriented programming sense, in that they only define a "class" of algorithms, rather than a specific algorithm.

2.1.3. Groupings

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

- transport-params-grouping

This grouping is presented in the following subsection.

2.1.3.1. 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*   identityref
  +-- key-exchange
  |  +-- key-exchange-alg*   identityref
  +-- encryption
  |  +-- encryption-alg*   identityref
  +-- mac
     +-- mac-alg*   identityref
```

Comments:

- This grouping is used by both the "ssh-client-grouping" and the "ssh-server-grouping" groupings defined in Section 3.1.2.1 and Section 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".

2.1.4. Protocol-accessible Nodes

The "ietf-ssh-common" module defines only "grouping" statements that are used by other modules to instantiate protocol-accessible nodes.

2.2. Example Usage

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

```
<!-- 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"
  xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">
  <host-key>
    <host-key-alg>algs:x509v3-rsa2048-sha256</host-key-alg>
    <host-key-alg>algs:ssh-rsa</host-key-alg>
  </host-key>
  <key-exchange>
    <key-exchange-alg>
      algs:diffie-hellman-group-exchange-sha256
    </key-exchange-alg>
  </key-exchange>
  <encryption>
    <encryption-alg>algs:aes256-ctr</encryption-alg>
    <encryption-alg>algs:aes192-ctr</encryption-alg>
    <encryption-alg>algs:aes128-ctr</encryption-alg>
    <encryption-alg>algs:aes256-cbc</encryption-alg>
    <encryption-alg>algs:aes192-cbc</encryption-alg>
    <encryption-alg>algs:aes128-cbc</encryption-alg>
  </encryption>
  <mac>
    <mac-alg>algs:hmac-sha2-256</mac-alg>
    <mac-alg>algs:hmac-sha2-512</mac-alg>
  </mac>
</transport-params>
```

2.3. YANG Module

This YANG module has normative references to [RFC4253], [RFC4344], [RFC4419], [RFC5656], [RFC6187], and [RFC6668].

<CODE BEGINS> file "ietf-ssh-common@2021-05-18.yang"

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

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

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

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

     Copyright (c) 2021 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Simplified
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

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

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
     'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
     are to be interpreted as described in BCP 14 (RFC 2119)
     (RFC 8174) when, and only when, they appear in all
     capitals, as shown here.";

  revision 2021-05-18 {
    description
      "Initial version";
    reference
      "RFC EEEE: YANG Groupings for SSH Clients and SSH Servers";
  }

  // Features

  feature ssh-ecc {
    description
      "Elliptic Curve Cryptography is supported for SSH.";
    reference
      "RFC 5656: Elliptic Curve Algorithm Integration in the
                 Secure Shell Transport Layer";
  }

  feature ssh-x509-certs {
    description
```

```
    "X.509v3 certificates are supported for SSH per RFC 6187.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell
               Authentication";
}

feature ssh-dh-group-exchange {
  description
    "Diffie-Hellman Group Exchange is supported for SSH.";
  reference
    "RFC 4419: Diffie-Hellman Group Exchange for the
               Secure Shell (SSH) Transport Layer Protocol";
}

feature ssh-ctr {
  description
    "SDCTR encryption mode is supported for SSH.";
  reference
    "RFC 4344: The Secure Shell (SSH) Transport Layer
               Encryption Modes";
}

feature ssh-sha2 {
  description
    "The SHA2 family of cryptographic hash functions is
     supported for SSH.";
  reference
    "FIPS PUB 180-4: Secure Hash Standard (SHS)";
}

// Identities

identity public-key-alg-base {
  description
    "Base identity used to identify public key algorithms.";
}

identity ssh-dss {
  base public-key-alg-base;
  description
    "Digital Signature Algorithm using SHA-1 as the
     hashing algorithm.";
  reference
    "RFC 4253:
       The Secure Shell (SSH) Transport Layer Protocol";
}

identity ssh-rsa {
  base public-key-alg-base;
  description
    "RSASSA-PKCS1-v1_5 signature scheme using SHA-1 as the
     hashing algorithm.";
  reference
    "RFC 4253:
       The Secure Shell (SSH) Transport Layer Protocol";
}

identity ecdsa-sha2-nistp256 {
```

```
  if-feature "ssh-ecc and ssh-sha2";
  base public-key-alg-base;
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
     nistp256 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
               Secure Shell Transport Layer";
}

identity ecdsa-sha2-nistp384 {
  if-feature "ssh-ecc and ssh-sha2";
  base public-key-alg-base;
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
     nistp384 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
               Secure Shell Transport Layer";
}

identity ecdsa-sha2-nistp521 {
  if-feature "ssh-ecc and ssh-sha2";
  base public-key-alg-base;
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
     nistp521 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
               Secure Shell Transport Layer";
}

identity x509v3-ssh-rsa {
  if-feature "ssh-x509-certs";
  base public-key-alg-base;
  description
    "RSASSA-PKCS1-v1_5 signature scheme using a public key stored
     in an X.509v3 certificate and using SHA-1 as the hashing
     algorithm.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell
               Authentication";
}

identity x509v3-rsa2048-sha256 {
  if-feature "ssh-x509-certs and ssh-sha2";
  base public-key-alg-base;
  description
    "RSASSA-PKCS1-v1_5 signature scheme using a public key stored
     in an X.509v3 certificate and using SHA-256 as the hashing
     algorithm.  RSA keys conveyed using this format MUST have a
     modulus of at least 2048 bits.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell
               Authentication";
}

identity x509v3-ecdsa-sha2-nistp256 {
```

```
  if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
  base public-key-alg-base;
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA)
     using the nistp256 curve with a public key stored in
     an X.509v3 certificate and using the SHA2 family of
     hashing algorithms.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell
               Authentication";
}

identity x509v3-ecdsa-sha2-nistp384 {
  if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
  base public-key-alg-base;
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA)
     using the nistp384 curve with a public key stored in
     an X.509v3 certificate and using the SHA2 family of
     hashing algorithms.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell
               Authentication";
}

identity x509v3-ecdsa-sha2-nistp521 {
  if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
  base public-key-alg-base;
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA)
     using the nistp521 curve with a public key stored in
     an X.509v3 certificate and using the SHA2 family of
     hashing algorithms.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell
               Authentication";
}

identity key-exchange-alg-base {
  description
    "Base identity used to identify key exchange algorithms.";
}

identity diffie-hellman-group14-sha1 {
  base key-exchange-alg-base;
  description
    "Diffie-Hellman key exchange with SHA-1 as HASH and
     Oakley Group 14 (2048-bit MODP Group).";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity diffie-hellman-group-exchange-sha1 {
  if-feature "ssh-dh-group-exchange";
  base key-exchange-alg-base;
  description
    "Diffie-Hellman Group and Key Exchange with SHA-1 as HASH.";
  reference
```

```
    "RFC 4419: Diffie-Hellman Group Exchange for the
               Secure Shell (SSH) Transport Layer Protocol";
}

identity diffie-hellman-group-exchange-sha256 {
  if-feature "ssh-dh-group-exchange and ssh-sha2";
  base key-exchange-alg-base;
  description
    "Diffie-Hellman Group and Key Exchange with SHA-256 as HASH.";
  reference
    "RFC 4419: Diffie-Hellman Group Exchange for the
               Secure Shell (SSH) Transport Layer Protocol";
}

identity ecdh-sha2-nistp256 {
  if-feature "ssh-ecc and ssh-sha2";
  base key-exchange-alg-base;
  description
    "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
     nistp256 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
               Secure Shell Transport Layer";
}

identity ecdh-sha2-nistp384 {
  if-feature "ssh-ecc and ssh-sha2";
  base key-exchange-alg-base;
  description
    "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
     nistp384 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
               Secure Shell Transport Layer";
}

identity ecdh-sha2-nistp521 {
  if-feature "ssh-ecc and ssh-sha2";
  base key-exchange-alg-base;
  description
    "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
     nistp521 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
               Secure Shell Transport Layer";
}

identity encryption-alg-base {
  description
    "Base identity used to identify encryption algorithms.";
}

identity triple-des-cbc {
  base encryption-alg-base;
  description
    "Three-key 3DES in CBC mode.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
```

```
}

identity aes128-cbc {
  base encryption-alg-base;
  description
    "AES in CBC mode, with a 128-bit key.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes192-cbc {
  base encryption-alg-base;
  description
    "AES in CBC mode, with a 192-bit key.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes256-cbc {
  base encryption-alg-base;
  description
    "AES in CBC mode, with a 256-bit key.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes128-ctr {
  if-feature "ssh-ctr";
  base encryption-alg-base;
  description
    "AES in SDCTR mode, with 128-bit key.";
  reference
    "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
               Modes";
}

identity aes192-ctr {
  if-feature "ssh-ctr";
  base encryption-alg-base;
  description
    "AES in SDCTR mode, with 192-bit key.";
  reference
    "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
               Modes";
}

identity aes256-ctr {
  if-feature "ssh-ctr";
  base encryption-alg-base;
  description
    "AES in SDCTR mode, with 256-bit key.";
  reference
    "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
       Modes";
}

identity mac-alg-base {
  description
```

```
    "Base identity used to identify message authentication
     code (MAC) algorithms.";
}

identity hmac-sha1 {
  base mac-alg-base;
  description
    "HMAC-SHA1";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity hmac-sha2-256 {
  if-feature "ssh-sha2";
  base mac-alg-base;
  description
    "HMAC-SHA2-256";
  reference
    "RFC 6668: SHA-2 Data Integrity Verification for the
               Secure Shell (SSH) Transport Layer Protocol";
}

identity hmac-sha2-512 {
  if-feature "ssh-sha2";
  base mac-alg-base;
  description
    "HMAC-SHA2-512";
  reference
    "RFC 6668: SHA-2 Data Integrity Verification for the
               Secure Shell (SSH) Transport Layer Protocol";
}

// 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 identityref {
        base public-key-alg-base;
      }
      ordered-by user;
      description
        "Acceptable host key algorithms in order of descending
         preference.  The configured host key algorithms should
         be compatible with the algorithm used by the configured
         private key.  Please see Section 5 of RFC EEEE for
         valid combinations.

         If this leaf-list is not configured (has zero elements)
         the acceptable host key algorithms are implementation-
         defined.";
      reference
```

```
          "RFC EEEE: YANG Groupings for SSH Clients and SSH Servers";
      }
    }
    container key-exchange {
      description
        "Parameters regarding key exchange.";
      leaf-list key-exchange-alg {
        type identityref {
          base key-exchange-alg-base;
        }
        ordered-by user;
        description
          "Acceptable key exchange algorithms in order of descending
           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 identityref {
          base encryption-alg-base;
        }
        ordered-by user;
        description
          "Acceptable encryption algorithms in order of descending
           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 identityref {
          base mac-alg-base;
        }
        ordered-by user;
        description
          "Acceptable MAC algorithms in order of descending
           preference.

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

<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 Examples (Section 3.2). 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-transport-params-config
  +-- ssh-client-keepalives
  +-- userauth-password
  +-- userauth-publickey
  +-- userauth-hostbased
  +-- userauth-none
```

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

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! {userauth-publickey}?
    |  |  +---u ks:local-or-keystore-asymmetric-key-grouping
    |  +-- password! {userauth-password}?
    |  |  +---u ct:password-grouping
    |  +-- hostbased! {userauth-hostbased}?
    |  |  +---u ks:local-or-keystore-asymmetric-key-grouping
    |  +-- none?          empty {userauth-none}?
    |  +-- certificate! {sshcmn:ssh-x509-certs}?
    |     +---u ks:local-or-keystore-end-entity-cert-with-key-groupi\
ng
    +-- server-authentication
    |  +-- ssh-host-keys!
    |  |  +---u ts:local-or-truststore-public-keys-grouping
    |  +-- ca-certs! {sshcmn:ssh-x509-certs}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- ee-certs! {sshcmn:ssh-x509-certs}?
    |     +---u ts:local-or-truststore-certs-grouping
    +-- transport-params {ssh-client-transport-params-config}?
    |  +---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 statement(s):
  - The "local-or-keystore-asymmetric-key-grouping" grouping is discussed in Section 2.1.3.4 of [I-D.ietf-netconf-keystore].
  - The "local-or-keystore-end-entity-cert-with-key-grouping" grouping is discussed in Section 2.1.3.6 of [I-D.ietf-netconf-keystore].
  - The "local-or-truststore-public-keys-grouping" grouping is discussed in Section 2.1.3.2 of [I-D.ietf-netconf-trust-anchors].
  - The "local-or-truststore-certs-grouping" grouping is discussed in Section 2.1.3.1 of [I-D.ietf-netconf-trust-anchors].

-The "transport-params-grouping" grouping is discussed in Section 2.1.3.1 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.

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 a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 2 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following configuration example uses local-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"
  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>
      <local-definition>
        <public-key-format>ct:ssh-public-key-format</public-key-form\
at>
        <public-key>base64encodedvalue==</public-key>
        <private-key-format>ct:rsa-private-key-format</private-key-f\
ormat>
        <cleartext-private-key>base64encodedvalue==</cleartext-priva\
te-key>
      </local-definition>
    </public-key>
  </client-identity>

  <!-- which host keys will this client trust -->
  <server-authentication>
    <ssh-host-keys>
      <local-definition>
        <public-key>
          <name>corp-fw1</name>
          <public-key-format>ct:ssh-public-key-format</public-key-fo\
rmat>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
        <public-key>
          <name>corp-fw2</name>
          <public-key-format>ct:ssh-public-key-format</public-key-fo\
rmat>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
      </local-definition>
    </ssh-host-keys>
    <ca-certs>
      <local-definition>
        <certificate>
          <name>Server Cert Issuer #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>Server Cert Issuer #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ca-certs>
    <ee-certs>
      <local-definition>
        <certificate>
```

```
          <name>My Application #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>My Application #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ee-certs>
  </server-authentication>

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

</ssh-client>
```

The following configuration example uses keystore-references for the client identity and 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>
    <!-- can an SSH client have move than one key?
    <public-key>
      <keystore-reference>ssh-rsa-key</keystore-reference>
    </public-key>
    -->
    <certificate>
      <keystore-reference>
        <asymmetric-key>ssh-rsa-key-with-cert</asymmetric-key>
        <certificate>ex-rsa-cert2</certificate>
      </keystore-reference>
    </certificate>
  </client-identity>

  <!-- which host-keys will this client trust -->
  <server-authentication>
    <ssh-host-keys>
      <truststore-reference>trusted-ssh-public-keys</truststore-refe\
rence>
    </ssh-host-keys>
    <ca-certs>
      <truststore-reference>trusted-server-ca-certs</truststore-refe\
rence>
    </ca-certs>
    <ee-certs>
      <truststore-reference>trusted-server-ee-certs</truststore-refe\
rence>
    </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 [I-D.ietf-netconf-trust-anchors], and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-ssh-client@2021-05-18.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 AAAA: YANG Data Types and Groupings for Cryptography";
  }

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

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

  import ietf-ssh-common {
    prefix sshcmn;
    revision-date 2021-05-18; // stable grouping definitions
    reference
      "RFC EEEE: YANG Groupings for SSH Clients and SSH Servers";
  }

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

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

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

     Copyright (c) 2021 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Simplified
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
```

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

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
   'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
   'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
   are to be interpreted as described in BCP 14 (RFC 2119)
   (RFC 8174) when, and only when, they appear in all
   capitals, as shown here.";

revision 2021-05-18 {
  description
    "Initial version";
  reference
    "RFC EEEE: YANG Groupings for SSH Clients and SSH Servers";
}

// Features

feature ssh-client-transport-params-config {
  description
    "SSH transport layer parameters are configurable on an SSH
     client.";
}

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

feature userauth-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 enable it to
     be disabled.";
  reference
    "RFC 4252:
      The Secure Shell (SSH) Authentication Protocol";
}

feature userauth-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 userauth-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 userauth-none {
  description
    "Indicates that the 'none' authentication type, per
     RFC 4252, is supported for client identification.";
  reference
    "RFC 4252:
      The Secure Shell (SSH) Authentication Protocol";
}

// Groupings

grouping ssh-client-grouping {
  description
    "A reusable grouping for configuring a 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 stack 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 RFC 4252,
       Section 5.2, 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 "userauth-publickey";
      presence
        "Indicates that publickey-based authentication has been
         configured.  This statement is present so the mandatory
         descendent 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 CCCC: A YANG Data Model for a Keystore";
  uses ks:local-or-keystore-asymmetric-key-grouping {
    refine "local-or-keystore/local/local-definition" {
      must 'public-key-format = "ct:ssh-public-key-format"';
    }
    refine "local-or-keystore/keystore/keystore-reference" {
      must 'deref(.)/../ks:public-key-format'
         + ' = "ct:ssh-public-key-format"';
    }
  }
}
container password {
  if-feature "userauth-password";
  presence
    "Indicates that password-based authentication has been
     configured.  This statement is present so the mandatory
     descendent 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 "userauth-hostbased";
  presence
    "Indicates that hostbased authentication is configured.
     This statement is present so the mandatory descendent
     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 CCCC: A YANG Data Model for a Keystore";
  uses ks:local-or-keystore-asymmetric-key-grouping {
    refine "local-or-keystore/local/local-definition" {
      must 'public-key-format = "ct:ssh-public-key-format"';
    }
    refine "local-or-keystore/keystore/keystore-reference" {
      must 'deref(.)/../ks:public-key-format'
         + ' = "ct:ssh-public-key-format"';
    }
  }
}
leaf none {
  if-feature "userauth-none";
  type empty;
  description
    "Indicates that '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 CCCC: A YANG Data Model for a Keystore";
    uses ks:local-or-keystore-end-entity-cert-with-key-grouping {
      refine "local-or-keystore/local/local-definition" {
        must 'public-key-format'
           + ' = "ct:subject-public-key-info-format"';
      }
      refine "local-or-keystore/keystore/keystore-reference"
           + "/asymmetric-key" {
        must '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 BBBB: A YANG Data Model for a Truststore";
    uses ts:local-or-truststore-public-keys-grouping {
      refine
        "local-or-truststore/local/local-definition/public-key" {
        must 'public-key-format = "ct:ssh-public-key-format"';
      }
      refine
        "local-or-truststore/truststore/truststore-reference" {
        must 'deref(.)/../*/ts:public-key-format'
           + ' = "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 certificate 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 BBBB: A YANG Data Model for a Truststore";
    uses ts:local-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 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 BBBB: A YANG Data Model for a Truststore";
    uses ts:local-or-truststore-certs-grouping;
  }
} // container server-authentication

container transport-params {
  nacm:default-deny-write;
  if-feature "ssh-client-transport-params-config";
  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 keep-alive policy, to proactively test
     the aliveness of the SSH server.  An unresponsive TLS
     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@ietf.org) 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 if
           no data has been received from the SSH server, a
           TLS-level message will be sent to test the
           aliveness of the SSH server.";
      }
      leaf max-attempts {
        type uint8;
        default "3";
        description
          "Sets the maximum number of sequential keep-alive
           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 Examples (Section 4.2). 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-transport-params-config
  +-- ssh-server-keepalives
  +-- local-users-supported
  +-- userauth-publickey
  +-- userauth-password
  +-- userauth-hostbased
  +-- userauth-none
```

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

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:local-or-keystore-asymmetric-key-grouping
    |        +--:(certificate)
    |           +-- certificate {sshcmn:ssh-x509-certs}?
    |              +---u ks:local-or-keystore-end-entity-cert-with-k\
ey-grouping
    +-- client-authentication
    |  +-- users {local-users-supported}?
    |  |  +-- user* [name]
    |  |     +-- name?          string
    |  |     +-- public-keys! {userauth-publickey}?
    |  |     |  +---u ts:local-or-truststore-public-keys-grouping
    |  |     +-- password?      ianach:crypt-hash
    |  |     |       {userauth-password}?
    |  |     +-- hostbased! {userauth-hostbased}?
    |  |     |  +---u ts:local-or-truststore-public-keys-grouping
    |  |     +-- none?          empty {userauth-none}?
    |  +-- ca-certs! {sshcmn:ssh-x509-certs}?
    |  |  +---u ts:local-or-truststore-certs-grouping
    |  +-- ee-certs! {sshcmn:ssh-x509-certs}?
    |     +---u ts:local-or-truststore-certs-grouping
    +-- transport-params {ssh-server-transport-params-config}?
    |  +---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 statement(s):
  -The "local-or-keystore-asymmetric-key-grouping" grouping is discussed in Section 2.1.3.4 of [I-D.ietf-netconf-keystore].

- The "local-or-keystore-end-entity-cert-with-key-grouping" grouping is discussed in Section 2.1.3.6 of [I-D.ietf-netconf-keystore].
- The "local-or-truststore-public-keys-grouping" grouping is discussed in Section 2.1.3.2 of [I-D.ietf-netconf-trust-anchors].
- The "local-or-truststore-certs-grouping" grouping is discussed in Section 2.1.3.1 of [I-D.ietf-netconf-trust-anchors].
- The "transport-params-grouping" grouping is discussed in Section 2.1.3.1 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.

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 a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 2 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following configuration example uses local-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"
  xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

  <!-- the host-key this SSH server will present -->
  <server-identity>
    <host-key>
      <name>my-pubkey-based-host-key</name>
      <public-key>
        <local-definition>
          <public-key-format>ct:ssh-public-key-format</public-key-fo\
rmat>
          <public-key>base64encodedvalue==</public-key>
          <private-key-format>ct:rsa-private-key-format</private-key\
-format>
          <cleartext-private-key>base64encodedvalue==</cleartext-pri\
vate-key>
        </local-definition>
      </public-key>
    </host-key>
    <host-key>
      <name>my-cert-based-host-key</name>
      <certificate>
        <local-definition>
          <public-key-format>ct:subject-public-key-info-format</publ\
ic-key-format>
          <public-key>base64encodedvalue==</public-key>
          <private-key-format>ct:rsa-private-key-format</private-key\
-format>
          <cleartext-private-key>base64encodedvalue==</cleartext-pri\
vate-key>
          <cert-data>base64encodedvalue==</cert-data>
        </local-definition>
      </certificate>
    </host-key>
  </server-identity>

  <!-- the client credentials this SSH server will trust -->
  <client-authentication>
    <users>
      <user>
        <name>mary</name>
        <password>$0$secret</password>
        <public-keys>
          <local-definition>
            <public-key>
              <name>User A</name>
              <public-key-format>ct:ssh-public-key-format</public-ke\
y-format>
              <public-key>base64encodedvalue==</public-key>
            </public-key>
            <public-key>
              <name>User B</name>
```

```
              <public-key-format>ct:ssh-public-key-format</public-ke\
y-format>
              <public-key>base64encodedvalue==</public-key>
            </public-key>
          </local-definition>
        </public-keys>
      </user>
    </users>
    <ca-certs>
      <local-definition>
        <certificate>
          <name>Identity Cert Issuer #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>Identity Cert Issuer #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ca-certs>
    <ee-certs>
      <local-definition>
        <certificate>
          <name>Application #1</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>Application #2</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
      </local-definition>
    </ee-certs>
  </client-authentication>

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

</ssh-server>
```

The following configuration example uses keystore-references for the server identity and 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"
  xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

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

  <!-- the client credentials this SSH server will trust -->
  <client-authentication>
    <users>
      <user>
        <name>mary</name>
        <password>$0$secret</password>
        <public-keys>
          <truststore-reference>SSH Public Keys for Application A</t\
ruststore-reference>
        </public-keys>
      </user>
    </users>
    <ca-certs>
      <truststore-reference>trusted-client-ca-certs</truststore-refe\
rence>
    </ca-certs>
    <ee-certs>
      <truststore-reference>trusted-client-ee-certs</truststore-refe\
rence>
    </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 [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore] and informative references to [RFC4253] and [RFC7317].

<CODE BEGINS> file "ietf-ssh-server@2021-05-18.yang"

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

  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 AAAA: YANG Data Types and Groupings for Cryptography";
  }

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

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

  import ietf-ssh-common {
    prefix sshcmn;
    revision-date 2021-05-18; // stable grouping definitions
    reference
      "RFC EEEE: YANG Groupings for SSH Clients and SSH Servers";
  }

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

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

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

     Copyright (c) 2021 IETF Trust and the persons identified
     as authors of the code. All rights reserved.
```

```
   Redistribution and use in source and binary forms, with
   or without modification, is permitted pursuant to, and
   subject to the license terms contained in, the Simplified
   BSD License set forth in Section 4.c of the IETF Trust's
   Legal Provisions Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

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

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
   'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
   'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
   are to be interpreted as described in BCP 14 (RFC 2119)
   (RFC 8174) when, and only when, they appear in all
   capitals, as shown here.";

 revision 2021-05-18 {
   description
     "Initial version";
   reference
     "RFC EEEE: YANG Groupings for SSH Clients and SSH Servers";
 }

 // Features

 feature ssh-server-transport-params-config {
   description
     "SSH transport layer parameters are configurable on an SSH
      server.";
 }

 feature ssh-server-keepalives {
   description
     "Per socket 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 userauth-publickey {
   description
     "Indicates that the 'publickey' authentication type, per
      RFC 4252, is supported.

      The 'publickey' authentication type is required by
      RFC 4252, but common implementations enable it to
      be disabled.";
   reference
     "RFC 4252:
       The Secure Shell (SSH) Authentication Protocol";
 }
```

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

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

feature userauth-none {
  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 a 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 stack 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 a SSH connection.";
    list host-key {
      key "name";
      min-elements 1;
      ordered-by user;
      description
        "An ordered list of host keys 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 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 CCCC: A YANG Data Model for a Keystore";
        uses ks:local-or-keystore-asymmetric-key-grouping {
          refine "local-or-keystore/local/local-definition" {
            must
              'public-key-format = "ct:ssh-public-key-format"';
          }
          refine "local-or-keystore/keystore/"
               + "keystore-reference" {
            must '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 CCCC: A YANG Data Model for a Keystore";
        uses
        ks:local-or-keystore-end-entity-cert-with-key-grouping {
          refine "local-or-keystore/local/local-definition" {
            must 'public-key-format'
               + ' = "ct:subject-public-key-info-format"';
          }
          refine "local-or-keystore/keystore/keystore-reference"
                 + "/asymmetric-key" {
            must '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 authenticate SSH clients.";
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 in 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 'user name' for the SSH client, as defined in
         the SSH_MSG_USERAUTH_REQUEST message in RFC 4253.";
    }
    container public-keys {
      if-feature "userauth-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 BBBB: A YANG Data Model for a Truststore";
      uses ts:local-or-truststore-public-keys-grouping {
        refine "local-or-truststore/local/local-definition"
               + "/public-key" {
          must 'public-key-format'
               + ' = "ct:ssh-public-key-format"';
        }
        refine "local-or-truststore/truststore/"
               + "truststore-reference" {
          must 'deref(.)/../*/ts:public-key-format'
               + ' = "ct:ssh-public-key-format"';
        }
      }
    }
    leaf password {
      if-feature "userauth-password";
      type ianach:crypt-hash;
```

```
      description
        "The password for this user.";
    }
    container hostbased {
      if-feature "userauth-hostbased";
      presence
        "Indicates that hostbased 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 4253: The Secure Shell (SSH) Transport Layer
         RFC BBBB: A YANG Data Model for a Truststore";
      uses ts:local-or-truststore-public-keys-grouping {
        refine "local-or-truststore/local/local-definition"
               + "/public-key" {
          must 'public-key-format'
               + ' = "ct:ssh-public-key-format"';
        }
        refine "local-or-truststore/truststore"
               + "/truststore-reference" {
          must 'deref(.)/../*/ts:public-key-format'
               + ' = "ct:ssh-public-key-format"';
        }
      }
    }
    leaf none {
      if-feature "userauth-none";
      type empty;
      description
        "Indicates that the 'none' method is supported.";
      reference
        "RFC 4252: The Secure Shell (SSH) Authentication
                   Protocol.";
    }
  }
}
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 certificate 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 BBBB: A YANG Data Model for a Truststore";
  uses ts:local-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 BBBB: A YANG Data Model for a Truststore";
    uses ts:local-or-truststore-certs-grouping;
  }
} // container client-authentication

container transport-params {
  nacm:default-deny-write;
  if-feature "ssh-server-transport-params-config";
  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 keep-alive policy, to proactively test
     the aliveness of the SSL client.  An unresponsive SSL
     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@ietf.org) 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
       if no data has been received from the SSL client,
       a SSL-level message will be sent to test the
       aliveness of the SSL client.";
  }
  leaf max-attempts {
    type uint8;
    default "3";
    description
```

```
          "Sets the maximum number of sequential keep-alive
           messages that can fail to obtain a response from
           the SSL client before assuming the SSL client is
           no longer alive.";
      }
    }
  } // grouping ssh-server-grouping

}
```

<CODE ENDS>

5. Security Considerations

5.1. The "ietf-ssh-common" YANG Module

The "ietf-ssh-common" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

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

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 does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

5.2. The "ietf-ssh-client" YANG Module

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]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

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

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.

Please be aware that this module uses the "key" and "private-key" nodes from the "ietf-crypto-types" module [I-D.ietf-netconf-crypto-types], where said nodes have the NACM extension "default-deny-all" set, thus preventing unrestricted read-access to the cleartext key values.

All of 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 consideration for such is not provided here.

5.3. The "ietf-ssh-server" YANG Module

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]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

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

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.

Please be aware that this module uses the "key" and "private-key" nodes from the "ietf-crypto-types" module [I-D.ietf-netconf-crypto-types], where said nodes have the NACM extension "default-deny-all" set, thus preventing unrestricted read-access to the cleartext key values.

All of 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 consideration for such is not provided here.

6. IANA Considerations

6.1. The "IETF XML" Registry

This document registers three URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

```
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

This document registers three YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

```
name:         ietf-ssh-common
namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-common
prefix:       sshcmn
reference:    RFC EEEE

name:         ietf-ssh-client
namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-client
prefix:       sshc
reference:    RFC EEEE

name:         ietf-ssh-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-server
prefix:       sshs
reference:    RFC EEEE
```

7. References

7.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "YANG Data Types and Groupings for Cryptography", Work in Progress, Internet-Draft, draft-ietf-netconf-crypto-types-19, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-crypto-types-19>.

[I-D.ietf-netconf-keystore] Watsen, K., "A YANG Data Model for a Keystore", Work in Progress, Internet-Draft, draft-ietf-

netconf-keystore-21, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-keystore-21>.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "A YANG Data Model for a Truststore", Work in Progress, Internet-Draft, draft-ietf-netconf-trust-anchors-14, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-trust-anchors-14>.

[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>.

[RFC4344] Bellare, M., Kohno, T., and C. Namprempre, "The Secure Shell (SSH) Transport Layer Encryption Modes", RFC 4344, DOI 10.17487/RFC4344, January 2006, <https://www.rfc-editor.org/info/rfc4344>.

[RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <https://www.rfc-editor.org/info/rfc4419>.

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

[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>.

[RFC6668] Bider, D. and M. Baushke, "SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol", RFC 6668, DOI 10.17487/RFC6668, July 2012, <https://www.rfc-editor.org/info/rfc6668>.

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

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

[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>.

7.2. Informative References

[I-D.ietf-netconf-http-client-server]
Watsen, K., "YANG Groupings for HTTP Clients and HTTP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-http-client-server-06, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-http-client-server-06>.

[I-D.ietf-netconf-netconf-client-server]
Watsen, K., "NETCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-netconf-client-server-22, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-netconf-client-server-22>.

[I-D.ietf-netconf-restconf-client-server]
Watsen, K., "RESTCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-client-server-22, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-restconf-client-server-22>.

[I-D.ietf-netconf-ssh-client-server]
Watsen, K., "YANG Groupings for SSH Clients and SSH Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-ssh-client-server-23, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-ssh-client-server-23>.

[I-D.ietf-netconf-tcp-client-server] Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-09, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-tcp-client-server-09>.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-23, 10 February 2021, <https://tools.ietf.org/html/draft-ietf-netconf-tls-client-server-23>.

[OPENSSH] Project, T. O., "OpenSSH", 2016, <http://www.openssh.com>.

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

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

Appendix A. Change Log

This section is to be removed before publishing as an RFC.

A.1. 00 to 01

*Noted that '0.0.0.0' and '::' might have special meanings.

*Renamed "keychain" to "keystore".

A.2. 01 to 02

*Removed the groupings 'listening-ssh-client-grouping' and 'listening-ssh-server-grouping'. Now modules only contain the transport-independent groupings.

*Simplified the "client-auth" part in the ietf-ssh-client module. It now inlines what it used to point to keystore for.

*Added cipher suites for various algorithms into new 'ietf-ssh-common' module.

A.3. 02 to 03

*Removed 'RESTRICTED' enum from 'password' leaf type.

*Added a 'must' statement to container 'server-auth' asserting that at least one of the various auth mechanisms must be specified.

*Fixed description statement for leaf 'trusted-ca-certs'.

A.4. 03 to 04

*Change title to "YANG Groupings for SSH Clients and SSH Servers"

*Added reference to RFC 6668

*Added RFC 8174 to Requirements Language Section.

*Enhanced description statement for ietf-ssh-server's "trusted-ca-certs" leaf.

*Added mandatory true to ietf-ssh-client's "client-auth" 'choice' statement.

*Changed the YANG prefix for module ietf-ssh-common from 'sshcom' to 'sshcmn'.

*Removed the compression algorithms as they are not commonly configurable in vendors' implementations.

*Updating descriptions in transport-params-grouping and the servers's usage of it.

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

*Updated YANG to use typedefs around leafrefs to common keystore paths

*Now inlines key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

*Merged changes from co-author.

A.6. 05 to 06

*Updated to use trust anchors from trust-anchors draft (was keystore draft)

*Now uses new keystore grouping enabling asymmetric key to be either locally defined or a reference to the keystore.

A.7. 06 to 07

*factored the ssh-[client|server]-groupings into more reusable groupings.

*added if-feature statements for the new "ssh-host-keys" and "x509-certificates" features defined in draft-ietf-netconf-trust-anchors.

A.8. 07 to 08

*Added a number of compatibility matrices to Section 5 (thanks Frank!)

*Clarified that any configured "host-key-alg" values need to be compatible with the configured private key.

A.9. 08 to 09

*Updated examples to reflect update to groupings defined in the keystore -09 draft.

*Add SSH keepalives features and groupings.

*Prefixed top-level SSH grouping nodes with 'ssh-' and support mashups.

*Updated copyright date, boilerplate template, affiliation, and folding algorithm.

A.10. 09 to 10

*Reformatted the YANG modules.

A.11. 10 to 11

*Reformatted lines causing folding to occur.

A.12. 11 to 12

*Collapsed all the inner groupings into the top-level grouping.

*Added a top-level "demux container" inside the top-level grouping.

*Added NACM statements and updated the Security Considerations section.

*Added "presence" statements on the "keepalive" containers, as was needed to address a validation error that appeared after adding the "must" statements into the NETCONF/RESTCONF client/server modules.

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

A.13. 12 to 13

*Removed the "demux containers", floating the nacm:default-deny-write to each descendent node, and adding a note to model designers regarding the potential need to add their own demux containers.

*Fixed a couple references (section 2 --> section 3)

*In the server model, replaced <client-cert-auth> with <client-authentication> and introduced 'local-or-external' choice.

A.14. 13 to 14

*Updated to reflect changes in trust-anchors drafts (e.g., s/trust-anchors/truststore/g + s/pinned.//)

A.15. 14 to 15

*Updated examples to reflect ietf-crypto-types change (e.g., identities --> enumerations)

*Updated "server-authentication" and "client-authentication" nodes from being a leaf of type "ts:host-keys-ref" or "ts:certificates-ref" to a container that uses "ts:local-or-truststore-host-keys-grouping" or "ts:local-or-truststore-certs-grouping".

A.16. 15 to 16

*Removed unnecessary if-feature statements in the -client and -server modules.

*Cleaned up some description statements in the -client and -server modules.

*Fixed a canonical ordering issue in ietf-ssh-common detected by new pyang.

A.17. 16 to 17

*Removed choice local-or-external by removing the 'external' case and flattening the 'local' case and adding a "local-users-supported" feature.

Updated examples to include the "-key-format" nodes.

*Augmented-in "must" expressions ensuring that locally-defined public-key-format are "ct:ssh-public-key-format" (must expr for ref'ed keys are TBD).

A.18. 17 to 18

*Removed leaf-list 'other' from ietf-ssh-server.

*Removed unused 'external-client-auth-supported' feature.

*Added features client-auth-password, client-auth-hostbased, and client-auth-none.

*Renamed 'host-key' to 'public-key' for when refering to 'publickey' based auth.

*Added new feature-protected 'hostbased' and 'none' to the 'user' node's config.

*Added new feature-protected 'hostbased' and 'none' to the 'client-identity' node's config.

*Updated examples to reflect new "bag" addition to truststore.

*Refined truststore/keystore groupings to ensure the key formats "must" be particular values.

*Switched to using truststore's new "public-key" bag (instead of separate "ssh-public-key" and "raw-public-key" bags.

*Updated client/server examples to cover ALL cases (local/ref x cert/raw-key/psk).

A.19. 18 to 19

*Updated the "keepalives" containers to address Michal Vasko's request to align with RFC 8071.

*Removed algorithm-mapping tables from the "SSH Common Model" section

*Removed 'algorithm' node from examples.

*Added feature "userauth-publickey"

*Removed "choice auth-type", as auth-types aren't exclusive.

*Renamed both "client-certs" and "server-certs" to "ee-certs"

*Switch "must" to assert the public-key-format is "subject-public-key-info-format" when certificates are used.

*Added a "Note to Reviewers" note to first page.

A.20. 19 to 20

*Added a "must 'public-key or password or hostbased or none or certificate'" statement to the "user" node in ietf-ssh-client

*Expanded "Data Model Overview section(s) [remove "wall" of tree diagrams].

*Moved the "ietf-ssh-common" module section to proceed the other two module sections.

*Updated the Security Considerations section.

A.21. 20 to 21

*Updated examples to reflect new "cleartext-" prefix in the crypto-types draft.

A.22. 21 to 22

*Cleaned up the SSH-client examples (i.e., removing FIXMEs)

*Fixed issues found by the SecDir review of the "keystore" draft.

*Updated the "ietf-ssh-client" module to use the new "password-grouping" grouping from the "crypto-types" module.

A.23. 22 to 23

*Addressed comments raised by YANG Doctor in the ct/ts/ks drafts.

A.24. 23 to 24

*Removed the 'supported-authentication-methods' from {grouping ssh-server-grouping}/client-authentication.

*Added XML-comment above examples explaining the reason for the unexepected top-most element's presence.

*Added RFC-references to various 'feature' statements.

*Renamed "credentials" to "authentication methods"

Renamed "client-auth-" to "userauth-*"

Renamed "client-identity-" to "userauth-*"

*Fixed nits found by YANG Doctor reviews.

*Aligned modules with `pyang -f` formatting.

*Added a 'Contributors' section.

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by first name): Alan Luchuk, Andy Bierman, Balazs Kovacs, Benoit Claise, Bert Wijnen, David Lamparter, Gary Wu, Juergen Schoenwaelder, Ladislav Lhotka, Liang Xia, Martin Bjoerklund, Mehmet Ersue, Michal Vasko, Phil Shafer, Radek Krejci, Sean Turner, Tom Petch.

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