NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2020

K. Watsen
Watsen Networks
G. Wu
Cisco Systems
L. Xia
Huawei

March 8, 2020

YANG Groupings for SSH Clients and SSH Servers
draft-ietf-netconf-ssh-client-server-18

Abstract

This document defines three YANG modules: the first defines groupings for a generic SSH client, the second defines groupings for a generic SSH server, and the third defines common identities and groupings used by both the client and the server. It is intended that these groupings will be used by applications using the SSH protocol.

Editorial Note (To be removed by RFC Editor)

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

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

- I-D.ietf-netconf-trust-anchors
- I-D.ietf-netconf-keystore

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-trust-anchors
- "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-keystore

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

- "2020-03-08" --> 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 a grouping for a generic SSH client, the second defines a grouping for a generic SSH server, and the third defines identities and groupings common to both the client and the 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.

2. Terminology

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

3. The SSH Client Model

3.1. Tree Diagram

This section provides a tree diagram [RFC8340] for the "ietf-ssh-client" module that does not have groupings expanded.


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

module: ietf-ssh-client

  grouping ssh-client-grouping
    +-- client-identity
    |  +-- username?            string
    |  +-- (auth-type)
    |     +--:(public-key)
    |     |  +-- public-key
    |     |     +---u ks:local-or-keystore-asymmetric-key-grouping
    |     +--:(password)
    |     |  +-- password?      string {client-identity-password}?
    |     +--:(hostbased)
    |     |  +-- hostbased {client-identity-hostbased}?
    |     |     +---u ks:local-or-keystore-asymmetric-key-grouping
    |     +--:(none)
    |     |  +-- none?          empty {client-identity-none}?
    |     +--:(certificate)
    |        +-- certificate {sshcmn:ssh-x509-certs}?
    |           +---u ks:local-or-keystore-end-entity-cert-with-key-\
grouping
```

```
      +-- 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
      |  +-- server-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
```

3.2. Example Usage

This section presents two examples showing the ssh-client-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 example configures the client identity using a local key:


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

<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>
        <algorithm>rsa2048</algorithm>
        <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>
          <private-key>base64encodedvalue==</private-key>
        </local-definition>
      </public-key>
    </client-identity>

    <!-- which host keys will this client trust -->
    <server-authentication>
      <ssh-host-keys>
        <local-definition> <!-- FIXME: float 'local-def' down to each?\
\ -->
          <!--<ssh-public-key>-->
          <public-key>
            <name>corp-fw1</name>
            <algorithm>secp256r1</algorithm>
            <public-key-format>ct:ssh-public-key-format</public-key-fo\
\rmat>
            <public-key>base64encodedvalue==</public-key>
            <!--
          </ssh-public-key>
          <ssh-public-key>
          -->
          </public-key>
          <public-key>
            <name>corp-fw2</name>
            <algorithm>secp256r1</algorithm>
            <public-key-format>ct:ssh-public-key-format</public-key-fo\
\rmat>
            <public-key>base64encodedvalue==</public-key>
            <!--</ssh-public-key>-->
          </public-key>
```



```
        </local-definition>
      </ssh-host-keys>
      <ca-certs>
        <local-definition>
          <!-- FIXME: float 'local-def' down to each?
          <certificate>
            <name>Server Cert Issuer #1</name>
            <cert>base64encodedvalue==</cert>
```

```
            </certificate>
            <certificate>
              <name>Server Cert Issuer #2</name>
              <cert>base64encodedvalue==</cert>
            </certificate>
            -->
            <cert>base64encodedvalue==</cert>
            <cert>base64encodedvalue==</cert>
          </local-definition>
        </ca-certs>
        <server-certs>
          <local-definition>
            <!-- FIXME: float 'local-def' down to each?
            <certificate>
              <name>My Application #1</name>
              <cert>base64encodedvalue==</cert>
            </certificate>
            <certificate>
              <name>My Application #2</name>
              <cert>base64encodedvalue==</cert>
            </certificate> -->
            <cert>base64encodedvalue==</cert>
            <cert>base64encodedvalue==</cert>
          </local-definition>
        </server-certs>
      </server-authentication>

      <transport-params>
        <host-key>
          <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>

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

  </ssh-client>
```

The following example configures the client identity using a key from the keystore:

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

  <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> <!-- FIXME: should 'ts-ref' be to bag or each ke\
  y? -->
        <truststore-reference>trusted-ssh-public-keys</truststore-refe\
  rence>
```

```
      </ssh-host-keys>
      <ca-certs> <!-- FIXME: should 'ts-ref' be to bag or each key? -->
        <truststore-reference>trusted-server-ca-certs</truststore-refe\
  rence>
      </ca-certs>
      <server-certs> <!-- FIXME: should 'ts-ref' be to bag or each key\
  ? -->
        <truststore-reference>trusted-server-ee-certs</truststore-refe\
  rence>
      </server-certs>
    </server-authentication>

    <transport-params>
      <host-key>
        <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>

    <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@2020-03-08.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: Common YANG Data Types 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 2020-03-08; // stable grouping definitions
    reference
      "RFC XXXX: 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.
```



```
      revision 2020-03-08 {
        description
          "Initial version";
        reference
          "RFC XXXX: 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 client-identity-password {
      description
        "Indicates that the 'password' authentication type
         is supported for client identification.";
    }
```



```
    feature client-identity-hostbased {
      description
        "Indicates that the 'hostbased' authentication type
         is supported for client identification.";
    }

    feature client-identity-none {
      description
        "Indicates that the 'none' authentication type is
         supported for client identification.";
    }

    // 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 descendent
         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 credentials used by the client to authenticate to
             the SSH server.";
          leaf username {
            type string;
            description
              "The username of this user.  This will be the username
               used, for instance, to log into an SSH server.";
          }
          choice auth-type {
            mandatory true;
            description
              "The authentication type.  What happens when more than
               one decendent is configured is undefined.  FIXME.";
            container public-key {
              // FIXME: no if-feature because required okay?
              description
```



```
                "A locally-defined or referenced asymmetric key
                 pair to be used for client identification.";
              reference
                "RFC ZZZZ: YANG Data Model for a Centralized
                           Keystore Mechanism";
              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 password {
              if-feature client-identity-password;
              nacm:default-deny-all;
```

```
          type string;
          description
            "A password to be used for client identification.";
        }
        container hostbased {
          if-feature client-identity-hostbased;
          description
            "A locally-defined or referenced asymmetric key
             pair to be used for host identification.";
          reference
            "RFC ZZZZ: YANG Data Model for a Centralized
                       Keystore Mechanism";
          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 client-identity-none;
          type empty;
          description
            "Indicates that 'none' algorithm is used for client
             identification.";
        }
```



```
        container certificate {
          if-feature "sshcmn:ssh-x509-certs";
          description
            "A locally-defined or referenced certificate
             to be used for client identification.";
          reference
            "RFC ZZZZ: YANG Data Model for a Centralized
                       Keystore Mechanism";
          uses
          ks:local-or-keystore-end-entity-cert-with-key-grouping {
            refine "local-or-keystore/local/local-definition" {
```

```
                must
                  'public-key-format = "ct:ssh-public-key-format"';
              } // FIXME: subject-public-key-info-format?
              refine "local-or-keystore/keystore/keystore-reference"
                     + "/asymmetric-key" {
                must 'deref(.)/../ks:public-key-format'
                     + ' = "ct:ssh-public-key-format"';
              } // FIXME: subject-public-key-info-format?
            }
          }
        }
      } // container client-identity

      container server-authentication {
        nacm:default-deny-write;
        must 'ssh-host-keys or ca-certs or server-certs';
        description
          "Specifies how the SSH client can authenticate SSH servers.
           Any combination of credentials is additive and unordered.";
        container ssh-host-keys {
          presence
            "Indicates that the client can authenticate servers
             using the configured SSH host keys.";
          description
            "A list 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 YYYY: YANG Data Model for Global Trust Anchors";
          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 client can authenticate servers
             using the configured trust anchor certificates.";
          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 YYYY: YANG Data Model for Global Trust Anchors";
          uses ts:local-or-truststore-certs-grouping;
            // Note: TS certs don't have a key-format...no test needed
        }
        container server-certs { // FIXME: rename to "ee-certs"?
          if-feature "sshcmn:ssh-x509-certs";
          presence
            "Indicates that the client can authenticate servers
             using the configured server certificates.";
          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
             server certificate.";
           reference
             "RFC YYYY: YANG Data Model for Global Trust Anchors";
          uses ts:local-or-truststore-certs-grouping;
            // Note: TS certs don't have a key-format...no test needed
        }
      } // 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 keepalives are enabled.";
          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.";
          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
    } // module ietf-ssh-client

    <CODE ENDS>
```

4. The SSH Server Model

4.1. Tree Diagram

This section provides a tree diagram [RFC8340] for the "ietf-ssh-server" module that does not have groupings expanded.

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

   module: ietf-ssh-server

     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
       |  +-- supported-authentication-methods
       |  |  +-- publickey?   empty
       |  |  +-- passsword?   empty {client-auth-password}?
       |  |  +-- hostbased?   empty {client-auth-hostbased}?
       |  |  +-- none?        empty {client-auth-none}?
       |  +-- users {client-auth-config-supported}?
       |  |  +-- user* [name]
       |  |     +-- name?          string
       |  |     +-- public-keys!
       |  |     |  +---u ts:local-or-truststore-public-keys-grouping
       |  |     +-- password?      ianach:crypt-hash
       |  |     |       {client-auth-password}?
       |  |     +-- hostbased! {client-auth-hostbased}?
       |  |     |  +---u ts:local-or-truststore-public-keys-grouping
       |  |     +-- none?          empty {client-auth-none}?
       |  +-- ca-certs!
       |  |       {client-auth-config-supported,sshcmn:ssh-x509-certs}?
       |  |  +---u ts:local-or-truststore-certs-grouping
       |  +-- client-certs!
       |          {client-auth-config-supported,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
```


4.2. Example Usage

This section presents two examples showing the ssh-server-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 example configures the server identity using a local key:

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

<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>
          <algorithm>rsa2048</algorithm>
          <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>
          <private-key>base64encodedvalue==</private-key>
```

```
          </local-definition>
        </public-key>
      </host-key>
      <host-key>
        <name>my-cert-based-host-key</name>
        <certificate>
          <local-definition>
            <algorithm>rsa2048</algorithm>
            <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>
            <private-key>base64encodedvalue==</private-key>
            <cert>base64encodedvalue==</cert>
```



```
          </local-definition>
        </certificate>
      </host-key>
    </server-identity>

    <!-- the client credentials this SSH server will trust -->
    <client-authentication>
      <supported-authentication-methods>
        <publickey/>
      </supported-authentication-methods>
      <users>
        <user>
          <name>mary</name>
          <password>$0$secret</password>
          <public-keys>
            <local-definition>
              <!--<ssh-public-key>-->
              <public-key>
                <name>User A</name>
                <algorithm>secp256r1</algorithm>
                <public-key-format>ct:ssh-public-key-format</public-ke\
y-format>
                <public-key>base64encodedvalue==</public-key>
                <!--</ssh-public-key>
              <ssh-public-key>-->
              </public-key>
```

```
              <public-key>
                <name>User B</name>
                <algorithm>secp256r1</algorithm>
                <public-key-format>ct:ssh-public-key-format</public-ke\
   y-format>
                <public-key>base64encodedvalue==</public-key>
              </public-key>
              <!--</ssh-public-key>-->
            </local-definition>
          </public-keys>
        </user>
      </users>
      <ca-certs>
        <local-definition>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
        </local-definition>
      </ca-certs>
      <client-certs>
        <local-definition>
          <cert>base64encodedvalue==</cert>
```



```
          <cert>base64encodedvalue==</cert>
          <cert>base64encodedvalue==</cert>
        </local-definition>
      </client-certs>
    </client-authentication>

    <transport-params>
      <host-key>
        <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>

  </ssh-server>
```

The following example configures the server identity using a key from the keystore:

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

<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>
        <supported-authentication-methods>
          <publickey/>
        </supported-authentication-methods>
        <users>
          <user>
            <name>mary</name>
            <password>$0$secret</password>
            <public-keys>
              <truststore-reference>SSH Public Keys for User A</truststo\
    re-reference>
            </public-keys>
          </user>
        </users>
        <ca-certs>
          <truststore-reference>trusted-client-ca-certs</truststore-refe\
    rence>
        </ca-certs>
        <client-certs>
          <truststore-reference>trusted-client-ee-certs</truststore-refe\
    rence>
        </client-certs>
      </client-authentication>

      <transport-params>
        <host-key>
          <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>

    </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@2020-03-08.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: Common YANG Data Types 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 2020-03-08; // stable grouping definitions
  reference
    "RFC XXXX: 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) 2019 IETF Trust and the persons identified
   as authors of the code. All rights reserved.

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

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

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

```
     revision 2020-03-08 {
       description
         "Initial version";
       reference
         "RFC XXXX: 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 client-auth-config-supported {
       description
         "Indicates that the configuration for how to authenticate
          clients can be configured herein, as opposed to in an
          application specific location.  That is, to support the
          consuming data models that prefer to place client
          authentication with client definitions, rather then
          in a data model principally concerned with configuring
          the transport.";
     }

     feature client-auth-password {
       description
         "Indicates that the 'password' authentication type
          is supported.";
     }

     feature client-auth-hostbased {
       description
         "Indicates that the 'hostbased' authentication type
          is supported.";
```

```
    }

    feature client-auth-none {
      description
        "Indicates that the 'none' authentication type is
         supported.";
```



```
    }

    // 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 descendent
         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 ZZZZ: YANG Data Model for a Centralized
                             Keystore Mechanism";
                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 ZZZZ: YANG Data Model for a Centralized
                             Keystore Mechanism";
                uses
                ks:local-or-keystore-end-entity-cert-with-key-grouping {
                  refine "local-or-keystore/local/local-definition" {
```

```
                must
                  'public-key-format = "ct:ssh-public-key-format"';
              } // FIXME: subject-public-key-info-format?
              refine "local-or-keystore/keystore/keystore-reference"
                     + "/asymmetric-key" {
                must 'deref(.)/../ks:public-key-format'
                     + ' = "ct:ssh-public-key-format"';
              } // FIXME: 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 supported-authentication-methods {
        description
```



```
          "Indicates which authentication methods the server
           supports.";
        leaf publickey {
          type empty;
          description
            "Indicates that the 'publickey' method is supported.
             Note that RFC 6187 X.509v3 Certificates for SSH uses
             the 'publickey' method name.";
          reference
            "RFC 4252: The Secure Shell (SSH) Authentication
                       Protocol.
             RFC 6187: X.509v3 Certificates for Secure Shell
                       Authentication.";
        }
        leaf passsword {
          if-feature client-auth-password;
          type empty;
          description
            "Indicates that the 'password' method is supported.";
          reference
            "RFC 4252: The Secure Shell (SSH) Authentication
```

```
                    Protocol.";
          }
          leaf hostbased {
            if-feature client-auth-hostbased;
            type empty;
            description
              "Indicates that the 'hostbased' method is supported.";
            reference
              "RFC 4252: The Secure Shell (SSH) Authentication
                         Protocol.";
          }
          leaf none {
            if-feature client-auth-none;
            type empty;
            description
              "Indicates that the 'none' method is supported.";
            reference
              "RFC 4252: The Secure Shell (SSH) Authentication
                         Protocol.";
          }
        }

        container users {
          if-feature "client-auth-config-supported";
          description
            "A list of locally configured users.";
          list user {
```



```
            key name;
            description
              "The list of local users configured on this device.";

            leaf name {
              type string;
              description
               "The user name string identifying this entry.";
            }

            container public-keys {
              // FIXME: no if-feature because required okay?
              presence
                "Indicates that the server can authenticate this
```

```
           user using any of the configured SSH public keys.";
         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 YYYY: 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 client-auth-password;
         type ianach:crypt-hash;
         description
           "The password for this user.";
       }

       container hostbased {
         if-feature client-auth-hostbased;
         presence
           "Indicates that the server can authenticate this
            user's 'host' using any of the configured SSH
```



```
            host keys.";
         description
           "A set of SSH host keys may be 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 YYYY: 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 client-auth-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 "client-auth-config-supported";
        if-feature "sshcmn:ssh-x509-certs";
        presence
          "Indicates that the SSH server can authenticate SSH
           clients using configured certificate authority (CA)
           certificates.";
        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 YYYY:
            YANG Data Model for Global Trust Anchors";
```

```
      uses ts:local-or-truststore-certs-grouping;
        // Note: TS certs don't have a key-format...no test needed
    }
    container client-certs {   // FIXME: plural too much?
      if-feature "client-auth-config-supported";
      if-feature "sshcmn:ssh-x509-certs";
      presence
        "Indicates that the SSH server can authenticate SSH
         clients using configured client certificates.";
      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 client certificate.";
      reference
        "RFC YYYY: A YANG Data Model for a Truststore";
      uses ts:local-or-truststore-certs-grouping;
        // Note: TS certs don't have a key-format...no test needed
    }
  } // 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 keepalives are enabled.";
    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.";
    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
    } // module ietf-ssh-server

    <CODE ENDS>
```

5. The SSH Common Model

The SSH common model presented in this section contains identities and groupings common to both SSH clients and SSH servers. The transport-params-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.

[I-D.ietf-netconf-crypto-types] defines six categories of cryptographic algorithms (hash-algorithm, symmetric-key-encryption-algorithm, mac-algorithm, asymmetric-key-encryption-algorithm, signature-algorithm, key-negotiation-algorithm) and lists several widely accepted algorithms for each of them. The SSH client and server models use one or more of these algorithms. The SSH common model includes four parameters for configuring its permitted SSH algorithms, which are: host-key-alg, key-exchange-alg, encryption-alg and mac-alg. The following tables are provided, in part, to define the subset of algorithms defined in the crypto-types model used by SSH and, in part, to ensure compatibility of configured SSH

cryptographic parameters for configuring its permitted SSH algorithms ("sshcmn" representing SSH common model, and "ct" representing crypto-types model which the SSH client/server model is based on):

sshcmn:host-key-alg	ct:signature-algorithm
dsa-sha1	dsa-sha1
rsa-pkcs1-sha1	rsa-pkcs1-sha1
rsa-pkcs1-sha256	rsa-pkcs1-sha256
rsa-pkcs1-sha512	rsa-pkcs1-sha512
ecdsa-secp256r1-sha256	ecdsa-secp256r1-sha256
ecdsa-secp384r1-sha384	ecdsa-secp384r1-sha384
ecdsa-secp521r1-sha512	ecdsa-secp521r1-sha512
x509v3-rsa-pkcs1-sha1	x509v3-rsa-pkcs1-sha1
x509v3-rsa2048-pkcs1-sha256	x509v3-rsa2048-pkcs1-sha1
x509v3-ecdsa-secp256r1-sha256	x509v3-ecdsa-secp256r1-sha256
x509v3-ecdsa-secp384r1-sha384	x509v3-ecdsa-secp384r1-sha384
x509v3-ecdsa-secp521r1-sha512	x509v3-ecdsa-secp521r1-sha512

Table 1 The SSH Host-key-alg Compatibility Matrix

sshcmn:key-exchange-alg	ct:key-negotiation-algorithm
diffie-hellman-group14-sha1	diffie-hellman-group14-sha1
diffie-hellman-group14-sha256	diffie-hellman-group14-sha256
diffie-hellman-group15-sha512	diffie-hellman-group15-sha512
diffie-hellman-group16-sha512	diffie-hellman-group16-sha512
diffie-hellman-group17-sha512	diffie-hellman-group17-sha512
diffie-hellman-group18-sha512	diffie-hellman-group18-sha512
ecdh-sha2-secp256r1	ecdh-sha2-secp256r1
ecdh-sha2-secp384r1	ecdh-sha2-secp384r1

Table 2 The SSH Key-exchange-alg Compatibility Matrix

sshcmn:encryption-alg	ct:symmetric-key-encryption-algorithm

aes-128-cbc	aes-128-cbc
aes-192-cbc	aes-192-cbc
aes-256-cbc	aes-256-cbc
aes-128-ctr	aes-128-ctr
aes-192-ctr	aes-192-ctr
aes-256-ctr	aes-256-ctr

Table 3 The SSH Encryption-alg Compatibility Matrix

sshcmn:mac-alg	ct:mac-algorithm
hmac-sha1	hmac-sha1
hmac-sha1-96	hmac-sha1-96
hmac-sha2-256	hmac-sha2-256
hmac-sha2-512	hmac-sha2-512

Table 4 The SSH Mac-alg Compatibility Matrix

As is seen in the tables above, the names of the "sshcmn" algorithms are all identical to the names of algorithms defined in [I-D.ietf-netconf-crypto-types]. While appearing to be redundant, it is important to realize that not all the algorithms defined in [I-D.ietf-netconf-crypto-types] are supported by SSH. That is, the algorithms supported by SSH are a subset of the algorithms defined in [I-D.ietf-netconf-crypto-types]. The algorithms used by SSH are redefined in this document in order to constrain the algorithms that may be selected to just the ones used by SSH.

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.

5.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-ssh-common" module.

```
module: ietf-ssh-common

  grouping transport-params-grouping
    +-- host-key
    |  +-- host-key-alg*   identityref
    +-- key-exchange
    |  +-- key-exchange-alg*   identityref
    +-- encryption
    |  +-- encryption-alg*   identityref
    +-- mac
       +-- mac-alg*   identityref
```


5.2. Example Usage

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

```
<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>
```

5.3. YANG Module

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

```
<CODE BEGINS> file "ietf-ssh-common@2020-03-08.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) 2019 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Simplified
     BSD License set forth in Section 4.c of the IETF Trust's
```

```
      Legal Provisions Relating to IETF Documents
      (https://trustee.ietf.org/license-info).

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

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

    revision 2020-03-08 {
      description
        "Initial version";
      reference
        "RFC XXXX: 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 XXXX for
           valid combinations.

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

```
          reference
            "RFC XXXX: 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>

6. Security Considerations

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

There are a number of data nodes defined in the YANG modules that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

*: All of the nodes defined by the grouping statement in both the "ietf-ssh-client" and "ietf-ssh-server" modules are sensitive to write operations. 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, all the nodes are protected the NACM extension "default-deny-write".

Some of the readable data nodes in the YANG modules may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

ssh-client-grouping/client-identity/: This subtree in the "ietf-ssh-client" module contains nodes that are additionally sensitive to read operations such that, in normal use cases, they should never be returned to a client. Specifically, the descendent nodes 'password', 'public-key/local-definition/private-key' and 'certificate/local-definition/private-key'. For this reason, all of these node are protected by the NACM extension "default-deny-all".

ssh-server-grouping/server-identity/: This subtree in the "ietf-ssh-server" module contains nodes that are additionally sensitive to read operations such that, in normal use cases, they should never be returned to a client. Specifically, the descendent nodes 'host-key/public-key/local-definition/private-key' and 'host-key/certificate/local-definition/private-key'. For this reason, both of these node are protected by the NACM extension "default-deny-all".

Some of the operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

*: The groupings defined in this document include "action" statements that come from groupings defined in [I-D.ietf-netconf-crypto-types]. Please consult that document for the security considerations of the "action" statements defined by the "grouping" statements defined in this document.

7. IANA Considerations

7.1. 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-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

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

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

7.2. 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-client
namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-client
prefix:       sshc
reference:    RFC XXXX

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

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

8. References

8.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K. and H. Wang, "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-13 (work in progress), November 2019.

[I-D.ietf-netconf-keystore]
Watsen, K., "A YANG Data Model for a Keystore", draft-ietf-netconf-keystore-15 (work in progress), November 2019.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "A YANG Data Model for a Truststore", draft-ietf-netconf-trust-anchors-08 (work in progress), November 2019.

[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>.

8.2. Informative References

[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>.

Appendix A. Change Log

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 "client-auth-config-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).

Acknowledgements

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

Authors' Addresses

Kent Watsen
Watsen Networks

EMail: kent+ietf@watsen.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com

Liang Xia
Huawei

EMail: frank.xialiang@huawei.com