Internet Engineering Task Force (IETF)
Request for Comments: 8177
Category: Standards Track
ISSN: 2070-1721

A. Lindem, Ed.
Cisco Systems
Y. Qu
Huawei
D. Yeung
Arrcus, Inc
I. Chen
Jabil
J. Zhang
Juniper Networks

June 2017

YANG Data Model for Key Chains

Abstract

This document describes the key chain YANG data model. Key chains are commonly used for routing protocol authentication and other applications requiring symmetric keys. A key chain is a list containing one or more elements containing a Key ID, key string, send/accept lifetimes, and the associated authentication or encryption algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, key strings and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated.

Status of This Memo

This is an Internet Standards Track document.

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

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8177.

Table of Contents

1. Introduction

This document describes the key chain YANG [YANG-1.1] data model. Key chains are commonly used for routing protocol authentication and other applications requiring symmetric keys. A key chain is a list containing one or more elements containing a Key ID, key string, send/accept lifetimes, and the associated authentication or encryption algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, key strings and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated.

In some applications, the protocols do not use the key chain element key directly, but rather a key derivation function is used to derive a short-lived key from the key chain element key (e.g., the master keys used in [TCP-AO]).

1.1. Requirements Notation

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 [KEYWORDS] [KEYWORDS-UPD] when, and only when, they appear in all capitals, as shown here.

1.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Section 3.3. The following tree notation is used.

- Brackets "[" and "]" enclose YANG list keys. These YANG list keys should not be confused with the key chain keys.
- Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- Abbreviations before data node names: "rw" means configuration (read-write), "ro" means state data (read-only), "-x" means RPC operations, and "-n" means notifications.
- Symbols after data node names: "?" means an optional node, "!" denotes a container with presence, and "*" denotes a "list" or "leaf-list".
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Problem Statement

This document describes a YANG [YANG-1.1] data model for key chains. Key chains have been implemented and deployed by a large percentage of network equipment vendors. Providing a standard YANG model will facilitate automated key distribution and non-disruptive key rollover. This will aid in tightening the security of the core routing infrastructure as recommended in [IAB-REPORT].

A key chain is a list containing one or more elements containing a Key ID, key string, send/accept lifetimes, and the associated authentication or encryption algorithm. A key chain can be used by any service or application requiring authentication or encryption using symmetric keys. In essence, the key chain is a reusable key policy that can be referenced wherever it is required. The key chain construct has been implemented by most networking vendors and deployed in many networks.

A conceptual representation of a crypto key table is described in [CRYPTO-KEYTABLE]. The crypto key table includes keys as well as their corresponding lifetimes and algorithms. Additionally, the key table includes key selection criteria and is designed for a deployment model where the details of the applications or services requiring authentication or encryption permeate into the key database. The YANG key chain model described herein doesn’t include key selection criteria or support this deployment model. At the same time, it does not preclude it. [YANG-CRYPTO-KEYTABLE] describes augmentations to the key chain YANG model in support of key selection criteria.

2.1. Applicability

Other YANG modules may reference ietf-key-chain YANG module key-chain names for authentication and encryption applications. A YANG type has been provided to facilitate reference to the key-chain name without having to specify the complete YANG XML Path Language (XPath) expression.

2.2. Graceful Key Rollover Using Key Chains

Key chains may be used to gracefully update the key string and/or algorithm used by an application for authentication or encryption. To achieve graceful key rollover, the receiver MAY accept all the

keys that have a valid accept lifetime, and the sender MAY send the key with the most recent send lifetime. One scenario for facilitating key rollover is to:

1. Distribute a key chain with a new key to all the routers or other network devices in the domain of that key chain. The new key's accept lifetime should be such that it is accepted during the key rollover period. The send lifetime should be a time in the future when it can be assured that all the routers in the domain of that key are upgraded. This will have no immediate impact on the keys used for transmission.

2. Assure that all the network devices have been updated with the updated key chain and that their system times are roughly synchronized. The system times of devices within an administrative domain are commonly synchronized (e.g., using the Network Time Protocol (NTP) [NTP-PROTO]). This also may be automated.

3. When the send lifetime of the new key becomes valid, the network devices within the domain of that key chain will use the new key for transmissions.

4. At some point in the future, a new key chain with the old key removed may be distributed to the network devices within the domain of the key chain. However, this may be deferred until the next key rollover. If this is done, the key chain will always include two keys: either the current and future key (during key rollovers) or the current and previous keys (between key rollovers).

Since the most recent send lifetime is defined as the one with the latest start-time, specification of "always" will prevent using the graceful key rollover technique described above. Other key configuration and usage scenarios are possible, but these are beyond the scope of this document.

3. Design of the Key Chain Model

The ietf-key-chain module contains a list of one or more keys indexed by a Key ID. For some applications (e.g., OSPFv3 [OSPFV3-AUTH]), the Key ID is used to identify the key chain key to be used. In addition to the Key ID, each key chain key includes a key string and a cryptographic algorithm. Optionally, the key chain keys include send/accept lifetimes. If the send/accept lifetime is unspecified, the key is always considered valid.

Note that different key values for transmission versus acceptance may be supported with multiple key chain elements. The key used for transmission will have a valid send-lifetime and invalid accept-lifetime (e.g., has an end-time equal to the start-time). The key used for acceptance will have a valid accept-lifetime and invalid send-lifetime.

Due to the differences in key chain implementations across various vendors, some of the data elements are optional. Finally, the crypto algorithm identities are provided for reuse when configuring legacy authentication and encryption not using key chains.

A key chain is identified by a unique name within the scope of the network device. The "key-chain-ref" typedef SHOULD be used by other YANG modules when they need to reference a configured key chain.

3.1. Key Chain Operational State

The key chain operational state is included in the same tree as key chain configuration consistent with Network Management Datastore Architecture [NMDA]. The timestamp of the last key chain modification is also maintained in the operational state. Additionally, the operational state includes an indication of whether or not a key chain key is valid for transmission or acceptance.

3.2. Key Chain Model Features

Features are used to handle differences between vendor implementations. For example, not all vendors support configuration of an acceptance tolerance or configuration of key strings in hexadecimal. They are also used to support security requirements (e.g., TCP-AO algorithms [TCP-AO-ALGORITHMS]) not yet implemented by vendors or implemented by only a single vendor.

It is common for an entity with sufficient permissions to read and store a device’s configuration, which would include the contents of this model. To avoid unnecessarily seeing and storing the keys in cleartext, this model provides the aes-key-wrap feature. More details are described in the Security Considerations (Section 5).

3.3. Key Chain Model Tree

```
   +--rw key-chains
      +--rw key-chain* [name]
      |  +--rw name                        string
      |  +--rw description?                string
      |  +--rw accept-tolerance {accept-tolerance}?
      |  |  +--rw duration?   uint32
      |  +--ro last-modified-timestamp?   yang:date-and-time
      |  +--rw key* [key-id]
      |     +--rw key-id                     uint64
      |     +--rw lifetime
      |     |  +--rw (lifetime)?
      |     |     +--:(send-and-accept-lifetime)
      |     |     |  +--rw send-accept-lifetime
      |     |     |     +--rw (lifetime)?
      |     |     |        +--:(always)
      |     |     |        |  +--rw always?            empty
      |     |     |        +--:(start-end-time)
      |     |     |           +--rw start-date-time?
      |     |     |           |       yang:date-and-time
      |     |     |           +--rw (end-time)?
      |     |     |              +--:(infinite)
      |     |     |              |  +--rw no-end-time?       empty
      |     |     |              +--:(duration)
      |     |     |              |  +--rw duration?          uint32
      |     |     |              +--:(end-date-time)
      |     |     |                 +--rw end-date-time?
      |     |     |                          yang:date-and-time
      |     |     +--:(independent-send-accept-lifetime)
      |     |        |   {independent-send-accept-lifetime}?
      |     |        +--rw send-lifetime
      |     |        |  +--rw (lifetime)?
      |     |        |     +--:(always)
      |     |        |     |  +--rw always?            empty
      |     |        |     +--:(start-end-time)
      |     |        |        +--rw start-date-time?
      |     |        |        |       yang:date-and-time
      |     |        |        +--rw (end-time)?
      |     |        |           +--:(infinite)
      |     |        |           |  +--rw no-end-time?       empty
      |     |        |           +--:(duration)
      |     |        |           |  +--rw duration?          uint32
      |     |        |           +--:(end-date-time)
      |     |        |              +--rw end-date-time?
      |     |        |                       yang:date-and-time
      |     |        +--rw accept-lifetime
      |     |           +--rw (lifetime)?
```

```
      |     |              +--:(always)
      |     |              |  +--rw always?            empty
      |     |              +--:(start-end-time)
      |     |                 +--rw start-date-time?
      |     |                 |       yang:date-and-time
      |     |                 +--rw (end-time)?
      |     |                    +--:(infinite)
      |     |                    |  +--rw no-end-time?       empty
      |     |                    +--:(duration)
      |     |                    |  +--rw duration?          uint32
      |     |                    +--:(end-date-time)
      |     |                       +--rw end-date-time?
      |     |                               yang:date-and-time
      |     +--rw crypto-algorithm identityref
      |     +--rw key-string
      |     |  +--rw (key-string-style)?
      |     |     +--:(keystring)
      |     |     |  +--rw keystring?            string
      |     |     +--:(hexadecimal) {hex-key-string}?
      |     |        +--rw hexadecimal-string?   yang:hex-string
      |     +--ro send-lifetime-active?     boolean
      |     +--ro accept-lifetime-active?   boolean
      +--rw aes-key-wrap {aes-key-wrap}?
         +--rw enable?   boolean
```

4. Key Chain YANG Model

```
<CODE BEGINS> file "ietf-key-chain@2017-06-15.yang"
module ietf-key-chain {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-key-chain";
  prefix key-chain;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-netconf-acm {
    prefix nacm;
  }

  organization
    "IETF RTGWG - Routing Area Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/group/rtgwg>
     WG List:  <mailto:rtgwg@ietf.org>

     Editor: Acee Lindem
             <mailto:acee@cisco.com>
```

```
                Yingzhen Qu
                <mailto:yingzhen.qu@huawei.com>
                Derek Yeung
                <mailto:derek@arrcus.com>
                Ing-Wher Chen
                <mailto:Ing-Wher_Chen@jabail.com>
                Jeffrey Zhang
                <mailto:zzhang@juniper.net>";

     description
       "This YANG module defines the generic configuration
        data for key chains.  It is intended that the module
        will be extended by vendors to define vendor-specific
        key chain configuration parameters.

        Copyright (c) 2017 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

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

        This version of this YANG module is part of RFC 8177;
        see the RFC itself for full legal notices.";

     reference "RFC 8177";

     revision 2017-06-15 {
       description
         "Initial RFC Revision";
       reference "RFC 8177: YANG Data Model for Key Chains";
     }

     feature hex-key-string {
       description
         "Support hexadecimal key string.";
     }

     feature accept-tolerance {
       description
         "Support the tolerance or acceptance limit.";
     }

     feature independent-send-accept-lifetime {
       description
```

```
        "Support for independent send and accept key lifetimes.";
    }

    feature crypto-hmac-sha-1-12 {
      description
        "Support for TCP HMAC-SHA-1 12-byte digest hack.";
    }

    feature cleartext {
      description
        "Support for cleartext algorithm.  Usage is
         NOT RECOMMENDED.";
    }

    feature aes-cmac-prf-128 {
      description
        "Support for AES Cipher-based Message Authentication
         Code Pseudorandom Function.";
    }

    feature aes-key-wrap {
      description
        "Support for Advanced Encryption Standard (AES) Key Wrap.";
    }

    feature replay-protection-only {
      description
        "Provide replay protection without any authentication
         as required by protocols such as Bidirectional
         Forwarding Detection (BFD).";
    }
    identity crypto-algorithm {
      description
        "Base identity of cryptographic algorithm options.";
    }

    identity hmac-sha-1-12 {
      base crypto-algorithm;
      if-feature "crypto-hmac-sha-1-12";
      description
        "The HMAC-SHA1-12 algorithm.";
    }

    identity aes-cmac-prf-128 {
      base crypto-algorithm;
      if-feature "aes-cmac-prf-128";
      description
        "The AES-CMAC-PRF-128 algorithm - required by
```

```
          RFC 5926 for TCP-AO key derivation functions.";
      }

      identity md5 {
        base crypto-algorithm;
        description
          "The MD5 algorithm.";
      }

      identity sha-1 {
        base crypto-algorithm;
        description
          "The SHA-1 algorithm.";
      }

      identity hmac-sha-1 {
        base crypto-algorithm;
        description
          "HMAC-SHA-1 authentication algorithm.";
      }

      identity hmac-sha-256 {
        base crypto-algorithm;
        description
          "HMAC-SHA-256 authentication algorithm.";
      }

      identity hmac-sha-384 {
        base crypto-algorithm;
        description
          "HMAC-SHA-384 authentication algorithm.";
      }

      identity hmac-sha-512 {
        base crypto-algorithm;
        description
          "HMAC-SHA-512 authentication algorithm.";
      }

      identity cleartext {
        base crypto-algorithm;
        if-feature "cleartext";
        description
          "cleartext.";
      }

      identity replay-protection-only {
        base crypto-algorithm;
```

```
      if-feature "replay-protection-only";
      description
        "Provide replay protection without any authentication as
         required by protocols such as Bidirectional Forwarding
         Detection (BFD).";
    }

    typedef key-chain-ref {
      type leafref {
        path
        "/key-chain:key-chains/key-chain:key-chain/key-chain:name";
      }
      description
        "This type is used by data models that need to reference
         configured key chains.";
    }

    grouping lifetime {
      description
        "Key lifetime specification.";
      choice lifetime {
        default "always";
        description
          "Options for specifying key accept or send lifetimes";
        case always {
          leaf always {
            type empty;
            description
              "Indicates key lifetime is always valid.";
          }
        }
        case start-end-time {
          leaf start-date-time {
            type yang:date-and-time;
            description
              "Start time.";
          }
          choice end-time {
            default "infinite";
            description
              "End-time setting.";
            case infinite {
              leaf no-end-time {
                type empty;
                description
                  "Indicates key lifetime end-time is infinite.";
              }
            }
```

```
            case duration {
              leaf duration {
                type uint32 {
                  range "1..2147483646";
                }
                units "seconds";
                description
                  "Key lifetime duration, in seconds";
              }
            }
            case end-date-time {
              leaf end-date-time {
                type yang:date-and-time;
                description
                  "End time.";
              }
            }
          }
        }
      }
    }

    container key-chains {
      description
        "All configured key-chains on the device.";
      list key-chain {
        key "name";
        description
          "List of key-chains.";
        leaf name {
          type string;
          description
            "Name of the key-chain.";
        }
        leaf description {
          type string;
          description
            "A description of the key-chain";
        }
        container accept-tolerance {
          if-feature "accept-tolerance";
          description
            "Tolerance for key lifetime acceptance (seconds).";
          leaf duration {
            type uint32;
            units "seconds";
            default "0";
            description
```

```
            "Tolerance range, in seconds.";
        }
      }
      leaf last-modified-timestamp {
        type yang:date-and-time;
        config false;
        description
          "Timestamp of the most recent update to the key-chain";
      }
      list key {
        key "key-id";
        description
          "Single key in key chain.";
        leaf key-id {
          type uint64;
          description
            "Numeric value uniquely identifying the key";
        }
        container lifetime {
          description
            "Specify a key's lifetime.";
          choice lifetime {
            description
              "Options for specification of send and accept
               lifetimes.";
            case send-and-accept-lifetime {
              description
                "Send and accept key have the same lifetime.";
              container send-accept-lifetime {
                description
                  "Single lifetime specification for both
                   send and accept lifetimes.";
                uses lifetime;
              }
            }
            case independent-send-accept-lifetime {
              if-feature "independent-send-accept-lifetime";
              description
                "Independent send and accept key lifetimes.";
              container send-lifetime {
                description
                  "Separate lifetime specification for send
                   lifetime.";
                uses lifetime;
              }
              container accept-lifetime {
                description
                  "Separate lifetime specification for accept
```

```
                lifetime.";
              uses lifetime;
            }
          }
        }
      }
      leaf crypto-algorithm {
        type identityref {
          base crypto-algorithm;
        }
        mandatory true;
        description
          "Cryptographic algorithm associated with key.";
      }
      container key-string {
        description
          "The key string.";
        nacm:default-deny-all;
        choice key-string-style {
          description
            "Key string styles";
           case keystring {
             leaf keystring {
              type string;
              description
                "Key string in ASCII format.";
            }
          }
          case hexadecimal {
            if-feature "hex-key-string";
            leaf hexadecimal-string {
              type yang:hex-string;
              description
                "Key in hexadecimal string format.  When compared
                 to ASCII, specification in hexadecimal affords
                 greater key entropy with the same number of
                 internal key-string octets.  Additionally, it
                 discourages usage of well-known words or
                 numbers.";
            }
          }
        }
      }
      leaf send-lifetime-active {
        type boolean;
        config false;
        description
          "Indicates if the send lifetime of the
```

```
                key-chain key is currently active.";
              }
            leaf accept-lifetime-active {
              type boolean;
              config false;
              description
                "Indicates if the accept lifetime of the
                 key-chain key is currently active.";
            }
          }
        }
        container aes-key-wrap {
          if-feature "aes-key-wrap";
          description
            "AES Key Wrap encryption for key-chain key-strings.  The
             encrypted key-strings are encoded as hexadecimal key
             strings using the hex-key-string leaf.";
          leaf enable {
            type boolean;
            default "false";
            description
              "Enable AES Key Wrap encryption.";
          }
        }
      }
    }
    <CODE ENDS>
```

5. Security Considerations

The YANG module defined in this document is designed to be accessed via network management protocols such as NETCONF [NETCONF] or RESTCONF [RESTCONF]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [NETCONF-SSH]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [TLS].

The NETCONF access control model [NETCONF-ACM] provides the means to restrict access for particular NETCONF or RESTCONF users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content. The key strings are not accessible by default, and NETCONF access control model [NETCONF-ACM] rules are required to configure or retrieve them.

When configured, the key strings can be encrypted using the AES Key Wrap algorithm [AES-KEY-WRAP]. The AES key-encryption key (KEK) is not included in the YANG model and must be set or derived independent of key chain configuration. When AES key encryption is used, the

hex-key-string feature is also required since the encrypted keys will contain characters that are not representable in the YANG string built-in type [YANG-1.1]. It is RECOMMENDED that key strings be encrypted using AES key encryption to prevent key chains from being retrieved and stored with the key strings in cleartext. This recommendation is independent of the access protection that is availed from the NETCONF access control model (NACM) [NETCONF-ACM].

The cleartext algorithm is included as a YANG feature. Usage is NOT RECOMMENDED except in cases where the application and device have no other alternative (e.g., a legacy network device that must authenticate packets at intervals of 10 milliseconds or less for many peers using Bidirectional Forwarding Detection [BFD]). Keys used with the cleartext algorithm are considered insecure and SHOULD NOT be reused with more secure algorithms.

Similarly, the MD5 and SHA-1 algorithms have been proven to be insecure ([Dobb96a], [Dobb96b], and [SHA-SEC-CON]), and usage is NOT RECOMMENDED. Usage should be confined to deployments where it is required for backward compatibility.

Implementations with keys provided via this model should store them using best current security practices.

6. IANA Considerations

This document registers a URI in the "IETF XML Registry" [XML-REGISTRY]. It follows the format in [XML-REGISTRY].

```
URI: urn:ietf:params:xml:ns:yang:ietf-key-chain
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
```

This document registers a YANG module in the "YANG Module Names" registry [YANG-1.0].

```
name: ietf-key-chain
namespace: urn:ietf:params:xml:ns:yang:ietf-key-chain
prefix: key-chain
reference: RFC 8177
```

7. References

7.1. Normative References

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

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

[NETCONF] 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, <http://www.rfc-editor.org/info/rfc6241>.

[NETCONF-ACM]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

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

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

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

[YANG-1.0]
Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

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

7.2. Informative References

[AES-KEY-WRAP]
Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <http://www.rfc-editor.org/info/rfc5649>.

[BFD] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <http://www.rfc-editor.org/info/rfc5880>.

[CRYPTO-KEYTABLE]
Housley, R., Polk, T., Hartman, S., and D. Zhang, "Database of Long-Lived Symmetric Cryptographic Keys", RFC 7210, DOI 10.17487/RFC7210, April 2014, <http://www.rfc-editor.org/info/rfc7210>.

[Dobb96a] Dobbertin, H., "Cryptanalysis of MD5 Compress", Technical Report Presented at the Rump Session of EuroCrypt ’96, May 1996.

[Dobb96b] Dobbertin, H., "The Status of MD5 After a Recent Attack", CryptoBytes, Vol. 2, No. 2, Summer 1996.

[IAB-REPORT]
Andersson, L., Davies, E., and L. Zhang, "Report from the IAB workshop on Unwanted Traffic March 9-10, 2006", RFC 4948, DOI 10.17487/RFC4948, August 2007, <http://www.rfc-editor.org/info/rfc4948>.

[NMDA] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", Work in Progress, draft-ietf-netmod-revised-datastores-02, May 2017.

[NTP-PROTO]
Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

[OSPFV3-AUTH]
Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, DOI 10.17487/RFC7166, March 2014, <http://www.rfc-editor.org/info/rfc7166>.

[SHA-SEC-CON]
Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <http://www.rfc-editor.org/info/rfc6194>.

[TCP-AO] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <http://www.rfc-editor.org/info/rfc5925>.

[TCP-AO-ALGORITHMS]
Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, DOI 10.17487/RFC5926, June 2010, <http://www.rfc-editor.org/info/rfc5926>.

[YANG-CRYPTO-KEYTABLE]
Chen, I., "YANG Data Model for RFC 7210 Key Table", Work in Progress, draft-chen-rtg-key-table-yang-00, March 2015.

Appendix A. Examples

A.1. Simple Key Chain with an Always Valid Single Key

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>keychain-no-end-time</name>
      <description>
        A key chain with a single key that is always valid for
        transmission and reception.
      </description>
      <key>
        <key-id>100</key-id>
        <lifetime>
          <send-accept-lifetime>
            <always/>
          </send-accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <keystring>keystring_in_ascii_100</keystring>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
</data>
```

A.2. Key Chain with Keys Having Different Lifetimes

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>keychain2</name>
      <description>
        A key chain where each key contains a different send time
        and accept time and a different algorithm illustrating
        algorithm agility.
      </description>
      <key>
        <key-id>35</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-01-01T00:00:00Z</start-date-time>
            <end-date-time>2017-02-01T00:00:00Z</end-date-time>
          </send-lifetime>
```

```
            <accept-lifetime>
              <start-date-time>2016-12-31T23:59:55Z</start-date-time>
              <end-date-time>2017-02-01T00:00:05Z</end-date-time>
            </accept-lifetime>
          </lifetime>
          <crypto-algorithm>hmac-sha-256</crypto-algorithm>
          <key-string>
            <keystring>keystring_in_ascii_35</keystring>
          </key-string>
        </key>
        <key>
          <key-id>36</key-id>
          <lifetime>
            <send-lifetime>
              <start-date-time>2017-02-01T00:00:00Z</start-date-time>
              <end-date-time>2017-03-01T00:00:00Z</end-date-time>
            </send-lifetime>
            <accept-lifetime>
              <start-date-time>2017-01-31T23:59:55Z</start-date-time>
              <end-date-time>2017-03-01T00:00:05Z</end-date-time>
            </accept-lifetime>
          </lifetime>
          <crypto-algorithm>hmac-sha-512</crypto-algorithm>
          <key-string>
            <hexadecimal-string>fe:ed:be:af:36</hexadecimal-string>
          </key-string>
        </key>
      </key-chain>
    </key-chains>
  </data>
```

A.3. Key Chain with Independent Send and Accept Lifetimes

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>keychain2</name>
      <description>
        A key chain where each key contains different send times
        and accept times.
      </description>
      <key>
        <key-id>35</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-01-01T00:00:00Z</start-date-time>
            <end-date-time>2017-02-01T00:00:00Z</end-date-time>
          </send-lifetime>
          <accept-lifetime>
            <start-date-time>2016-12-31T23:59:55Z</start-date-time>
            <end-date-time>2017-02-01T00:00:05Z</end-date-time>
          </accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <keystring>keystring_in_ascii_35</keystring>
        </key-string>
      </key>
      <key>
        <key-id>36</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-02-01T00:00:00Z</start-date-time>
            <end-date-time>2017-03-01T00:00:00Z</end-date-time>
          </send-lifetime>
          <accept-lifetime>
            <start-date-time>2017-01-31T23:59:55Z</start-date-time>
            <end-date-time>2017-03-01T00:00:05Z</end-date-time>
          </accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <hexadecimal-string>fe:ed:be:af:36</hexadecimal-string>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
</data>
```

Contributors

Yi Yang
SockRate

Email: yi.yang@sockrate.com

Acknowledgments

Thanks to Brian Weis for fruitful discussions on security requirements.

Thanks to Ines Robles for Routing Directorate QA review comments.

Thanks to Ladislav Lhotka for YANG Doctor comments.

Thanks to Martin Bjorklund for additional YANG Doctor comments.

Thanks to Tom Petch for comments during IETF last call.

Thanks to Matthew Miller for comments made during the Gen-ART review.

Thanks to Vincent Roca for comments made during the Security Directorate review.

Thanks to Warren Kumari, Ben Campbell, Adam Roach, and Benoit Claise for comments received during the IESG review.

Authors' Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
United States of America

Email: acee@cisco.com

Yingzhen Qu
Huawei

Email: yingzhen.qu@huawei.com

Derek Yeung
Arrcus, Inc

Email: derek@arrcus.com

Ing-Wher Chen
Jabil

Email: Ing-Wher_Chen@jabil.com

Jeffrey Zhang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
United States of America

Email: zzhang@juniper.net