Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 31, 2017

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

April 29, 2017

Routing Key Chain YANG Data Model
draft-ietf-rtgwg-yang-key-chain-24.txt

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 of elements each containing a key string, send lifetime, accept lifetime, and algorithm (authentication or encryption). 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 Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 31, 2017.

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 of elements each containing a key string, send lifetime, accept lifetime, and algorithm (authentication or encryption). 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 [RFC-KEYWORDS].

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" state data (read-only), "-x" RPC operations, and "-n" notifications.
- Symbols after data node names: "?" means an optional node, "!" 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 (":").
- 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 also includes keys as well as their corresponding lifetimes and algorithms. Additionally, the key table includes key selection criteria and envisions 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. The draft [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) selector.

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 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 key chain will using 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 sending 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 of security requirements (e.g., TCP-AO Algorithms [TCP-AO-ALGORITHMS]) not yet implemented by vendors or 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 clear-text, this model provides the aes-key-wrap feature. More details are described in 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@2017-04-18.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 RTG (Routing) Working Group";
  contact
    "Acee Lindem - acee@cisco.com";
  description
    "This YANG module defines the generic configuration
     data for key-chain. 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 XXXX; see
     the RFC itself for full legal notices.";

  revision 2017-04-18 {
    description
      "Initial RFC Revision";
    reference "RFC XXXX: A YANG Data Model for key-chain";
  }
```

```
  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 clear-text {
    description
      "Support for clear-text algorithm. Usage is
       NOT RECOMMENDED.";
  }

  feature aes-cmac-prf-128 {
    description
      "Support for AES Cipher based Message Authentication
       Code Pseudo Random 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 clear-text {
      base crypto-algorithm;
      if-feature "clear-text";
      description
        "Clear text.";
    }

    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.";
              }
            }
          }
        }
      }
    }
    grouping key-common {
      description
        "Key-chain key data nodes common to
         configuration and state.";
    }

    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 Mode [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 clear text. This recommendation is independent of the access protection that is availed from the NETCONF Access Control Model (NACM) [NETCONF-ACM].

The clear-text 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 clear-text 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]. Following the format in [XML-REGISTRY], the following registration is requested to be made:

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 XXXX

7. Contributors

Contributors' Addresses

Yi Yang
SockRate

Email: yi.yang@sockrate.com

8. References

8.1. Normative References

[NETCONF] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[NETCONF-ACM] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC-KEYWORDS] Bradner, S., "Key words for use in RFC's to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[XML-REGISTRY] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[YANG-1.0]
Bjorklund, M., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[YANG-1.1]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.

8.2. Informative References

[AES-KEY-WRAP]
Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 5649, August 2009.

[BFD] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, June 2010.

[CRYPTO-KEYTABLE]
Housley, R., Polk, T., Hartman, S., and D. Zhang, "Table of Cryptographic Keys", RFC 7210, April 2014.

[Dobb96a] Dobbertin, H., "Cryptanalysis of MD5 Compress", Technical Report (Presented at the RUMP Session of EuroCrypt 1996), 2 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, August 2007.

[NETCONF-SSH]
Wasserman, M., "NETCONF over SSH", RFC 6242, June 2011.

[NMDA] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watson, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01.txt (work in progress), March 2017.

[NTP-PROTO]
Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[OSPFV3-AUTH]
Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, March 2014.

[RESTCONF]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, January 2017.

[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, February 2011.

[TCP-AO] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[TCP-AO-ALGORITHMS]
Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol", RFC 5246, August 2008.

[YANG-CRYPTO-KEYTABLE]
Chen, I., "YANG Data Model for RFC 7210 Key Table", draft-chen-rtg-key-table-yang-00.txt (work in progress), November 2015.

Appendix A. Examples

A.1. Simple Key Chain with 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 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 time
        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>
```

Appendix B. Acknowledgments

The RFC text was produced using Marshall Rose's xml2rfc tool.

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
USA

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
USA

Email: zzhang@juniper.net