

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 31, 2015

A. Lindem, Ed.  
Y. Qu  
D. Yeung  
Cisco Systems  
I. Chen  
Ericsson  
J. Zhang  
Juniper Networks  
Y. Yang  
Cisco Systems  
February 27, 2015

**Key Chain YANG Data Model**  
**draft-acee-rtg-yang-key-chain-01.txt**

**Abstract**

This document describes the key chain YANG data model. A key chain is a list of elements each containing a key, send lifetime, accept lifetime, and algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, keys and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated. Key chains are commonly used for routing protocol authentication and other applications. 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.

**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 August 31, 2015.

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                             |  |                    |
|-----------------------------|--|--------------------|
| <a href="#">1.</a>          | Introduction . . . . .                           | <a href="#">2</a>  |
| <a href="#">1.1.</a>        | Requirements Notation . . . . .                  | <a href="#">3</a>  |
| <a href="#">2.</a>          | Problem Statement . . . . .                      | <a href="#">3</a>  |
| <a href="#">2.1.</a>        | Graceful Key Rollover using Key Chains . . . . . | <a href="#">3</a>  |
| <a href="#">3.</a>          | Design of the Key Chain Model . . . . .          | <a href="#">4</a>  |
| <a href="#">4.</a>          | Key Chain YANG Model . . . . .                   | <a href="#">6</a>  |
| <a href="#">5.</a>          | Security Considerations . . . . .                | <a href="#">11</a> |
| <a href="#">6.</a>          | IANA Considerations . . . . .                    | <a href="#">11</a> |
| <a href="#">7.</a>          | References . . . . .                             | <a href="#">11</a> |
| <a href="#">7.1.</a>        | Normative References . . . . .                   | <a href="#">11</a> |
| <a href="#">7.2.</a>        | Informative References . . . . .                 | <a href="#">12</a> |
| <a href="#">Appendix A.</a> | Acknowledgments . . . . .                        | <a href="#">12</a> |
|                             | Authors' Addresses . . . . .                     | <a href="#">12</a> |

## [1.](#) Introduction

This document describes the key chain YANG data model. A key chain is a list of elements each containing a key, send lifetime, accept lifetime, and algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, keys and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated. Key chains are commonly used for routing protocol authentication and other applications. 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.



### **1.1. Requirements Notation**

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC-KEYWORDS](#)].

## **2. Problem Statement**

This document describes a YANG [[YANG](#)] 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 of containing one or more elements containing a Key ID, key, send/accept lifetimes, and the associated authentication or encryption algorithm. A conceptual representation of a crypto key table is described in [[CRYPTO-KEYTABLE](#)]. The key chain model presented herein represents a practical implementation of the crypto key table. However, the key selection is left to the applications requiring authentication or encryption. This is more inline with the current operational model.

### **2.1. Graceful Key Rollover using Key Chains**

Key chains may be used to gracefully update the key and/or algorithm used by an application for authentication or encryption. This MAY be accomplished by accepting all the keys that have a valid accept lifetime and sending 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 start sending the new key.
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).

### 3. Design of the Key Chain Model

The ietf-keychain 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 element to be used. In addition to the Key-ID, each key chain element includes a key-string. Optionally, the key chain entries include send/accept lifetimes and a cryptographic algorithm. If the send/accept lifetime is unspecified, the key is always considered valid.

Note that asymmetric keys, i.e., a different key value used for transmission versus acceptance, may be supported with multiple key chain elements where the accept-lifetime or send-lifetime is not valid (e.g., has an end-time equal to the start-time).

Due to the differences in key chain implementations across various vendors, some of the data elements are optional. Additionally, the key-chain is made a grouping so that an implementation could support scoping other than at the global level.

```

+--rw key-chains* [name]
  +--rw name                string
  +--rw accept-tolerance {accept-tolerance}?
  | +--rw duration?         uint32
  +--rw key* [key-id]
    +--rw key-id             uint64
    +--rw key-string
    | +--rw (key-string-style)?
    | +--:(keystring)
    | | +--rw keystring?      string
    | +--:(hexadecimal) {hex-key-string}?
    | +--rw hexadecimal-string? yang:hex-string
  +--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 {cryptographic-algorithm}?
|         |   +--rw (algorithm)?
|         |   |   +--:(hmac-sha1-12)
|         |   |   |   +--rw hmac-sha1-12?  empty
|         |   |   +--:(hmac-sha1-20)
|         |   |   |   +--rw hmac-sha1-20?  empty
|         |   |   +--:(md5)
|         |   |   |   +--rw md5?           empty

```





```

+--:(sha-1)
|  +--rw sha-1?          empty
+--:(hmac-sha-1)
|  +--rw hmac-sha-1?     empty
+--:(hmac-sha-256)
|  +--rw hmac-sha-256?   empty
+--:(hmac-sha-384)
|  +--rw hmac-sha-384?   empty
+--:(hmac-sha-512)
|  +--rw hmac-sha-512?   empty

```

#### 4. Key Chain YANG Model

```

module ietf-key-chain {
  namespace "urn:ietf:params:xml:ns:yang:ietf-key-chain";
  // replace with IANA namespace when assigned
  prefix key-chain;

  import ietf-yang-types {
    prefix "yang";
  }

  organization
    "Cisco Systems
     170 West Tasman Drive
     San Jose, CA 95134-1706
     USA";
  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.
    ";

  revision 2015-02-24 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: A YANG Data Model for key-chain";
  }

  feature cryptographic-algorithm {
    description
      "Support cryptographic algorithm.";
  }
}
```



```
}

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

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

feature independent-send-accept-lifetime {
  description
    "Support for independent send and accept key lifetimes.";
}

grouping lifetime {
  description
    "Key lifetime specification.";
  choice lifetime {
    default always;
    case always {
      leaf always {
        type empty;
      }
      description
        "Key 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
          "Never expires.";
      }
    }
    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-chain {
    description
        "Grouping for one key-chain.";

    leaf name {
        type string;
        description "Name of the key-chain.";
    }

    container accept-tolerance {
        if-feature accept-tolerance;
        leaf duration {
            type uint32;
            units seconds;
            default "0";
            description
                "Tolerance range, in seconds.";
        }
    }

    list key {
        key "key-id";
        description "One key.";
        leaf key-id {
            type uint64;
            description "Key id.";
        }
        container key-string {
            description "The key string.";
            choice key-string-style {
                description
```



```
        "Key string styles";
    case keystack {
        leaf keystack {
            type string;
        }
    }
    case hexadecimal {
        if-feature hex-key-string;
        leaf hexadecimal-string {
            type yang:hex-string;
            description
                "Hexadecimal string.";
        }
    }
}

container lifetime {
    description "Specify a key's lifetime.";
    choice lifetime {
        case send-and-accept-lifetime {
            description
                "Send and accept key have the same lifetime.";
            container send-accept-lifetime {
                uses lifetime;
            }
        }
        case independent-send-accept-lifetime {
            if-feature independent-send-accept-lifetime;
            description
                "Independent send and accept key lifetimes.";
            container send-lifetime {
                uses lifetime;
            }
            container accept-lifetime {
                uses lifetime;
            }
        }
    }
}

container crypto-algorithm {
    if-feature cryptographic-algorithm;
    choice algorithm {
        case hmac-sha1-12 {
            leaf hmac-sha1-12 {
                type empty;
                description "The HMAC-SHA1-12 algorithm.";
            }
        }
    }
}
```





```
    }
    case hmac-sha1-20 {
      leaf hmac-sha1-20 {
        type empty;
        description "The HMAC-SHA1-20 algorithm.";
      }
    }
    case md5 {
      leaf md5 {
        type empty;
        description "The MD5 algorithm.";
      }
    }
    case sha-1 {
      leaf sha-1 {
        type empty;
        description "The SHA-1 algorithm.";
      }
    }
    case hmac-sha-1 {
      leaf hmac-sha-1 {
        type empty;
        description "HMAC-SHA-1 authentication algorithm.";
      }
    }
    case hmac-sha-256 {
      leaf hmac-sha-256 {
        type empty;
        description "HMAC-SHA-256 authentication algorithm.";
      }
    }
    case hmac-sha-384 {
      leaf hmac-sha-384 {
        type empty;
        description "HMAC-SHA-384 authentication algorithm.";
      }
    }
    case hmac-sha-512 {
      leaf hmac-sha-512 {
        type empty;
        description "HMAC-SHA-512 authentication algorithm.";
      }
    }
  }
  description "The crypto algorithm used.";
}
}
```



```
list key-chains {  
  key "name";  
  description  
    "A key-chain is a sequence of keys that are collectively  
    managed for authentication.";  
  uses key-chain;  
}  
}
```

## 5. Security Considerations

This document enables the automated distribution of industry standard key chains using the NETCONF [\[NETCONF\]](#) protocol. As such, the security considerations for the NETCONF protocol are applicable. Given that the key chains themselves are sensitive data, it is RECOMMENDED that the NETCONF communication channel be encrypted. One way to do accomplish this would be to invoke and run NETCONF over SSH as described in [\[NETCONF-SSH\]](#).

## 6. IANA Considerations

This document registers a URI in the IETF XML registry [\[XML-REGISTRY\]](#). Following the format in [RFC 3688](#), 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 [\[YANG\]](#).

name: ietf-acl namespace: urn:ietf:params:xml:ns:yang:ietf-key-chain prefix: ietf-key-chain reference: RFC XXXX

## 7. References

### 7.1. Normative References

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



## [NETCONF-SSH]

Wasserman, M., "Using NETCONF Protocol over Secure Shell (SSH)", [RFC 6242](#), June 2011.

## [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]

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

## [7.2. Informative References](#)

## [CRYPTO-KEYTABLE]

Housley, R., Polk, T., Hartman, S., and D. Zhang, "Table of Cryptographic Keys", [RFC 7210](#), April 2014.

## [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.

## [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.

## [Appendix A. Acknowledgments](#)

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

## Authors' Addresses



Acee Lindem (editor)  
Cisco Systems  
301 Midenhall Way  
Cary, NC 27513  
USA

Email: [acee@cisco.com](mailto:acee@cisco.com)

Yingzhen Qu  
Cisco Systems  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Email: [yiqu@cisco.com](mailto:yiqu@cisco.com)

Derek Yeung  
Cisco Systems  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Email: [myeung@cisco.com](mailto:myeung@cisco.com)

Ing-Wher Chen  
Ericsson

Email: [ing-wher.chen@ericsson.com](mailto:ing-wher.chen@ericsson.com)

Jeffrey Zhang  
Juniper Networks  
10 Technology Park Drive  
Westford, MA 01886  
USA

Email: [zzhang@juniper.net](mailto:zzhang@juniper.net)





Yi Yang  
Cisco Systems  
7025 Kit Creek Road  
Research Triangle Park, NC 27709  
USA  
  
Email: [yiya@cisco.com](mailto:yiya@cisco.com)