6TiSCH Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 1, 2018

M. Richardson
Sandelman Software Works

August 28, 2017

Minimal Security rekeying mechanism for 6TiSCH
draft-richardson-6tisch-minimal-rekey-02

Abstract

This draft describes a mechanism to rekey the networks used by 6TISCH nodes. It leverages the security association created during an enrollment protocol. The rekey mechanism permits incremental deployment of new sets of keys, followed by a rollover to a new 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 March 1, 2018.

Table of Contents

1. Introduction

6TiSCH networks of nodes often use a pair of keys, K1/K2 to authenticate beacons (K1), encrypt broadcast traffic (K1) and encrypt unicast traffic (K2). These keys need to occasionally be refreshed for a number of reasons:

- cryptographic hygiene: the keys must be replaced before the ASN roles over or there could be repeated use of the same key.

- to remove nodes from the group: replacing the keys excludes any nodes that are suspect, or which are known to have left the network

- to recover short-addresses: if the JRC is running out of short (2-byte) addresses, it can rekey the network in order to garbage collect the set of addresses.

This protocol uses the CoMI [I-D.ietf-core-comi] to present the set of 127 key pairs.

In addition to providing for rekey, this protocol includes access to the allocated short-address.

2. Terminology

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 [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

The reader is expected to be familiar with the terms and concepts defined in [I-D.ietf-6tisch-terminology], [RFC7252], [I-D.ietf-core-object-security], and [I-D.ietf-anima-bootstrapping-keyinfra].

3. Tree diagram notation

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- Abbreviations before data node names: "rw" (read-write) represents configuration data and "ro" (read-only) represents state data.
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and 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.

4. An approach to rekeying

Rekeying of the network requires that all nodes be updated with the new keys. This can take time as the network is constrained, and this management traffic is not highest priority.

The JRC must reach out to all nodes that it is aware of. As the JRC has originally provided the keys via either zero-touch [I-D.ietf-6tisch-dtsecurity-secure-join] or [I-D.ietf-6tisch-minimal-security] protocol, and in each case, the JRC assigned the short-address to the node, so it knows about all the nodes.

The data model presented in this document provides for up to 127 K1/K2 keys, as each key requires a secKeyId, which is allocated from a 255-element palette provides by [IEEE8021542015]. Keys are to be updated in pairs, and the pairs are associated in the following way: the K1 key is always the odd numbered key (1,3,5), and the K2 key is the even numbered key that follows (2,4,6). A secKeyId value of 0 is invalid, and the secKeyId value of 255 is unused in this process.

Nodes MAY support up to all 127 key pair slots, but MUST support a minimum of 6 keys (3 slot-pairs). When fewer than 127 are supported, the node MUST support secKeyId values from 1 to 254 in a sparse array fashion.

A particular key slot-pair is considered active, and this model provides a mechanism to query and also to explicitely set the active pair.

Nodes decrypt any packets for which they have keys, but MUST continue to send using only the keypair which is considered active. Receipt of a packet which is encrypted (or authenticated in the case of a broadcast) with a secKeyId larger (taking consideration that secKeyId wraps at 254) than the active slot-pair causes the node to change active slot pairs.

This mechanism permits the JRC to provision new keys into all the nodes while the network continues to use the existing keys. When the JRC is certain that all (or enough) nodes have been provisioned with the new keys, then the JRC causes a packet to be sent using the new key. This can be the JRC sending the next Enhanced Beacon or unicast traffic using the new key if the JRC is also a regular member of the LLN. In the likely case that the JRC has no direct connection to the LLN, then the JRC updates the active key to the new key pair using a CoMI message.

The frame goes out with the new keys, and upon receipt (and decryption) of the new frame all receiving nodes will switch to the new active key. Beacons or unicast traffic leaving those nodes will then update additional peers, and the network will switch over in a flood-fill fashion.

((EDNOTE: do we need an example?))

5. YANG models

5.1. Tree diagram

A diagram of the two YANG modules looks like:

```
1700  module: ietf-6tisch-symmetric-keying
1701      +--rw ietf6tischkeypairs* [counter]
1702      |  +--rw counter           uint16
1703      |  +--rw ietf6tischkey1
1704      |  |  +--rw secKeyDescriptor
1705      |  |  |  +--rw secKey?   binary
1706      |  |  +--rw secKeyIndex?        uint8
1707      |  +--rw ietf6tischkey2
1708      |     +--rw secKeyDescriptor
1709      |     |  +--rw secKey?   binary
1710      |     +--rw secKeyIndex?        uint8
1711      +--ro secKeyUsage
1712      |  +--ro txPacketsSent?       uint32
1713      |  +--ro rxPacketsSuccess?    uint32
1714      |  +--ro rxPacketsReceived?   uint32
1715      +--rw newKey?               binary

         rpcs:
1716       +---x installNextKey

1717  module: ietf-6tisch-short-address
1718      +--ro ietf6shortaddresses
1719         +--ro shortaddress    binary
1720         +--ro validuntil      uint32
1721         +--ro effectiveat?    uint32
```

Figure 1: Tree diagrams of two rekey modules

5.2. YANG model for keys

```
module ietf-6tisch-symmetric-keying {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-6tisch-symmetric-keying";
  prefix "ietf6keys";

  //import ietf-yang-types { prefix yang; }
  //import ietf-inet-types { prefix inet; }

  organization
   "IETF 6tisch Working Group";
```

```
  contact
   "WG Web:   <http://tools.ietf.org/wg/6tisch/>
    WG List:  <mailto:6tisch@ietf.org>
    Author:   Michael Richardson
              <mailto:mcr+ietf@sandelman.ca>";

  description
    "This module defines the format for a set of network-wide 802.15.4
    keys used in 6tisch networks.  There are 128 sets of key pairs,
    with one keypair (K1) used to authenticate (and sometimes encrypt)
    multicast traffic, and another keypair (K2) used to encrypt unicast
    traffic.  The 128 key pairs are numbered by the (lower) odd
    keyindex, which otherwise is a 0-255 value.  Keyindex 0 is
    not valid.  This module is a partial expression of the tables in
    https://mentor.ieee.org/802.15/dcn/15/15-15-0106-07-0mag-security-section-
pictures.pdf.
To read and write the key pairs, a monotonically increasing counter is added. A
new key pair must be added with current_counter = last_counter+1. The current
specification allows overwriting of earlier key pairs. It is up to the server
to remove old key pairs, such that only the last three (two) pairs are stored
and visible to the client.";

  revision "2017-03-01" {
    description
     "Initial version";
    reference
     "RFC XXXX: 6tisch minimal security";
  }

  // list of key pairs
  list ietf6tischkeypairs {
  key counter;
  description
    "a list of key pairs with unique index: counter.";
  leaf  counter {
  type uint16{
     range "0..256";  // for the moment 256 items
    }
  mandatory "true";
  description
    "unique reference to the key pair for client access.";
  }  // counter

  // key descriptor for FIRST part of pair
  container ietf6tischkey1 {
    description
      "A voucher that can be used to assign one or more
       devices to an owner.";
```

```
// this container is pretty empty, a leaf would do the job.

    container secKeyDescriptor {
  // I assume this needs to be extended, why else a container?
      description
```

```
        "This container describes the details of a
           specific cipher key";
      leaf secKey {
        type binary;
        description "The actual encryption key.
          This value is write only, and is not returned in a
          read, or returns all zeroes.";
      } // secKey
    }  // secKeyDescriptor

    // leaf secKeyIdMode is always 1, not described here.
    leaf secKeyIndex {
      type uint8;
      description
        "The keyIndex for this keySet.
            A number between 1 and 255.";
      reference
        "IEEE802.15.4";
    } // secKeyIndex
 } // ietf6tischkey1

// key descriptor for SECOND part of pair
  container ietf6tischkey2 {
    description
      "A voucher that can be used to assign one or more
       devices to an owner.";
    container secKeyDescriptor {
  // I assume this needs to be extended, why else a container?
      description
        "This container describes the details of a
           specific cipher key";
      leaf secKey {
        type binary;
        description "The actual encryption key.
          This value is write only, and is not returned in a
          read, or returns all zeroes.";
      } // secKey
    }  // secKeyDescriptor

    // leaf secKeyIdMode is always 1, not described here.
    leaf secKeyIndex {
      type uint8;
      description
        "The keyIndex for this keySet.
           A number between 1 and 255.";
      reference
        "IEEE802.15.4";
    } // secKeyIndex
```

```
   } // ietf6tischkey2
 } //ietf6tischkeypairs

// the usage is over all pairs
    container secKeyUsage {
    config false; // cannot be set by client
    description
      "statistics of sent and received packets.";
      leaf txPacketsSent {
        type uint32;
        description "Number of packets sent with this key.";
      } // txPacketsSent
      leaf rxPacketsSuccess {
        type uint32;
        description "Number of packets received with this key that were
                     successfully decrypted and authenticated.";
      }// rxPacketsSuccess
      leaf rxPacketsReceived {
        type uint32;
        description "Number of packets received with this key, both
            successfully received, and unsuccessfully.";
      } // rxPacketsReceived

    } // secKeyUsage

 // setting new key, and validation of new key
     leaf newKey{
        type binary;
        description
        "new key value to be set by client.";
     }  // newKey
     rpc installNextKey{
         description
         "Client informs server that newKey is to be
         used as current key.";
     }  // installNextKey


  } // module ietf-6tisch-symmetric-keying
```

5.3. YANG model for short-address

```
   module ietf-6tisch-short-address {
     yang-version 1.1;

     namespace
       "urn:ietf:params:xml:ns:yang:ietf-6tisch-short-address";
     prefix "ietf6shortaddr";
```

```
     //import ietf-yang-types { prefix yang; }
     //import ietf-inet-types { prefix inet; }

     organization
      "IETF 6tisch Working Group";

     contact
      "WG Web:   <http://tools.ietf.org/wg/6tisch/>
       WG List:  <mailto:6tisch@ietf.org>
       Author:   Michael Richardson
                 <mailto:mcr+ietf@sandelman.ca>";

     description
      "This module defines an interface to set and interrogate
       the short (16-bit) layer-2 address used in 802.15.4
       TSCH mode networks.  The short addresses are used
       in L2 frames to save space.  A lifetime is included
       in terms of TSCH Absolute Slot Number, which acts
       as a monotonically increasing clock.  ";

     revision "2017-03-01" {
       description
        "Initial version";
       reference
        "RFC XXXX: 6tisch minimal security";
     }

     // top-level container
     container ietf6shortaddresses {
       config false;
       description
         "A 16-bit short address for use by the node.";

       leaf shortaddress {
         type binary{
            length 1..2;}
         mandatory true;
         description
           "The two byte short address to be set.";
       }
       leaf validuntil {
         type uint32;
         mandatory true;
         description "The Absolute Slot Number/256 at which
                      the address ceases to be valid.";
       }

       leaf effectiveat {
```

```
        type uint32;
        description "The Absolute Slot Number/256 at which
                     time the address was originally set.
                     This is a read-only attribute that
                     records the ASN when the shortaddress
                     element was last written or updated.";
      }
    }
  }
```

6. Security of CoMI link

The CoMI resources presented here are protected by OSCOAP ([I-D.ietf-core-object-security]), secured using the EDHOC connection used for joining. A unique application key is generated using an additional key generation process with the unique label "6tisch-rekey".

7. Rekey of master connection

Should the OSCOAP connection need to be rekeyed, a new EDHOC process will be necessary. This will need access to trusted authentication keys, either the PSK used from a one-touch process, or the locally significant domain certificates installed during a zero-touch process.

8. Privacy Considerations

The rekey protocol itself runs over a network encrypted with the K2 key. The end to end protocol from JRC to node is also encrypted using OSCOAP, so the keys are not visible, nor is the keying traffic distinguished in anyway to an observer.

As the secKeyId is not confidential in the underlying 802.15.4 frames, an observer can determine what sets of keys are in use, and when a rekey is activated by observing the change in the secKeyId.

The absolute value of the monitonically increasing secKeyId could provide some information as to the age of the network.

9. Security Considerations

This protocol permits the underlying network keys to be set. Access to all of the portions of this interface MUST be restricted to an ultimately trusted peer, such as the JRC.

An implementation SHOULD not permit reading the network keys. Those fields should be write-only.

The OSCOAP security for this interface is initialized by a join mechanism, and so depends upon the initial credentials provided to the node. The initial network keys would have been provided during the join process; this protocol permits them to be updated.

10. IANA Considerations

This document allocates a SID number for the YANG model. There is no IANA action required for this document.

11. Acknowledgments

12. References

12.1. Normative References

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-01 (work in progress), July 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-04 (work in progress), July 2017.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

12.2. Informative References

[I-D.ietf-6tisch-6top-protocol]
Wang, Q., Vilajosana, X., and T. Watteyne, "6top Protocol (6P)", draft-ietf-6tisch-6top-protocol-07 (work in progress), June 2017.

[I-D.ietf-6tisch-dtsecurity-secure-join]
Richardson, M., "6tisch Secure Join protocol", draft-ietf-6tisch-dtsecurity-secure-join-01 (work in progress), February 2017.

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-03 (work in progress), June 2017.

[I-D.ietf-6tisch-terminology]
Palattella, M., Thubert, P., Watteyne, T., and Q. Wang, "Terminology in IPv6 over the TSCH mode of IEEE 802.15.4e", draft-ietf-6tisch-terminology-09 (work in progress), June 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-07 (work in progress), July 2017.

[IEEE8021542015]
IEEE standard for Information Technology, ., "IEEE Std 802.15.4-2015 Standard for Low-Rate Wireless Personal Area Networks (WPANs)", 2015.

Appendix A. Example

In the examples below, a new key value is set in the server example.com; followed by the setting of the new key value as the current key value. The SID values of new Key and installNextKey are 1715 and 1716 respectively. The corresponding base64 values are: ez and e0 respectively.

The setting of the new key value is done with the PUT request with the binary value 1234567890.

```
   PUT coap://example.com/c/ez
      (Content-Format :application/yang-value+cbor)
   h'1234567890'

   RES: 2.01 Created

   Payload in CBOR:
   45            # bytes(5)
      1234567890 # "\x124Vx\x90"
```

Consecutively, the RPC is invoked with a POST method to validate the new key value.

```
   POST coap://example.com/c/e0
      (Content-Format :application/yang-value+cbor)

   RES: 2.05 Content
```

The client can query how many TX packets have been received. The SID of secKeyUsage/txPacketsSent is 1712, corresponding with base64 ew.

```
   GET coap://example.com/c/ew

   RES: 2.05 Content (Content-Format :application/yang-value+cbor)
   3

   Payload in CBOR:
   03 # unsigned(3)
```

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca