R. Housley
Vigil Security
T. Polk
NIST
S. Hartman
Painless Security
D. Zhang
Huawei

Database of Long-Lived Symmetric Cryptographic Keys
<**draft-ietf-karp-crypto-key-table-10.txt**>

Abstract

   This document specifies the information contained in a conceptual
   database of long-lived cryptographic keys used by many different
   routing protocols for message security. The database is designed to
   support both manual and automated key management. In addition to
   describing the schema for the database, this document describes the
   operations that can be performed on the database as well as the
   requirements for the routing protocols that wish to use the database.
   In many typical scenarios, the protocols do not directly use the
   long-lived key, but rather a key derivation function is used to
   derive a short-lived key from a long-lived key.

Status of this Memo

1. Introduction

   This document specifies the information that needs to be included in
   a database of long-lived cryptographic keys in order to key the
   cryptographic authentication of routing protocols. This conceptual
   database is designed to separate protocol-specific aspects from both
   manual and automated key management. The intent is to allow many
   different implementation approaches to the specified cryptographic
   key database, while simplifying specification and heterogeneous
   deployments. This conceptual database avoids the need to build
   knowledge of any security protocol into key management protocols. It
   minimizes protocol-specific knowledge in operational/management
   interfaces, but it constrains where that knowledge can appear.
   Textual conventions are provided for the representation of keys and
   other identifiers. These conventions should be used when presenting
   keys and identifiers to operational/management interfaces or reading
   keys/identifiers from these interfaces. This satisfies the
   operational requirement that all implementations represent the keys
   and key identifiers in the same way so that cross-vendor
   configuration instructions can be provided.

   Routing protocols can employ the services of more generic security
   protocols such as TCP-AO [RFC5925]. Implementations of routing
   protocols may need to supply keys to databases specific to these
   security protocols as the associated entries in this document's
   conceptual database are manipulated.

   In many instances, the long-lived keys are not used directly in
   security protocols, but rather a key derivation function is used to
   derive short-lived keys from the long-lived key in the database. In
   other instances, security protocols will directly use the long-lived
   key from the database. The database design supports both use cases.

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 2119 [RFC2119].

2. Conceptual Database Structure

The database is characterized as a table, where each row represents a
single long-lived symmetric cryptographic key. Normally, each key
should only have one row. Only in the (hopefully) very rare cases
where a key is used for more than one purpose, or where the same key
is used with multiple key derivation functions (KDFs) will multiple
rows contain the same key value. The columns in the table represent
the key value and attributes of the key.

To accommodate manual key management, the format of the fields has
been purposefully chosen to allow updates with a plain text editor
and to provide equivalent display on multiple systems.

The columns that the table consists of are listed as follows:

AdminKeyName
    The AdminKeyName field contains a human-readable string meant
    to identify the key for the user. Implementations can use this
    field to uniquely identify rows in the key table. The same
    string can be used on the local system and peer systems, but
    this is not required. Routing protocols do not make use of this
    string; they use the LocalKeyName and the PeerKeyName. However,
    if these strings are to be used as protocol elements in other
    protocols or otherwise transferred between systems, they will
    need to follow the requirements of section 5.1.

LocalKeyName
    The LocalKeyName field contains a string identifying the key.
    It can be used to retrieve the key in the local database when
    received in a message. As discussed in Section 4, the
    protocol defines the form of this field. For example, many
    routing protocols restrict the format of their key names to
    integers that can be represented in 16 or 32 bits. Typically
    this field does not contain data in human character sets
    requiring internationalization. If there ever are any routing
    Protocols with key names requiring internationalization, those
    specifications need to address issues of canonicalization and
    normalization so that key names can be compared using binary
    comparison.

PeerKeyName
    PeerKeyName is the name of the key used by the local system in
    an outgoing message. For unicast communication, the PeerKeyName
    of a key on a system matches the LocalKeyName of the identical
    key that is maintained on one or multiple peer systems. Similar
    to LocalKeyName, a protocol defines the form of this identifier
    and will often restrict it to be an integer. For group keys,
    the protocol will typically require this field be an empty
    string as the sending and the receiving key names need to be
    the same.

Peers
    Typically for unicast keys, this field lists the peer systems
    that have this key in their database. For group keys this field
    names the groups for which the key is appropriate. For example,
    this might name a routing area for a multicast routing
    protocol. Formally, this field provides a protocol-specific set
    of restrictions on the scope in which the key is appropriate.
    The format of the identifiers in the Peers field is specified
    by the protocol.

Interfaces
    The Interfaces field identifies the set of physical and/or
    virtual interfaces for which it is appropriate to use this key.
    When the long-lived value in the Key field is intended for use
    on any interface, this field is set to "all". The interfaces
    field consists of a set of strings; the form of these strings
    is specified by the implementation and is independent of the
    protocol in question. Protocols may require support for the
    interfaces field or may indicate that support for constraining
    keys based on interface is not required. As an example, TCP-AO
    implementations are unlikely to make the decision of what
    interface to use prior to key selection. In this case, the
    implementations are expected to use the same keying material
    across all of the interfaces and then require the "all"
    setting.

Protocol
    The Protocol field identifies a single routing protocol where
    this key may be used to provide cryptographic protection. This
    specification establishes a registry for this field; the
    registry also specifies the format of the following field,
    ProtocolSpecificInfo, for each registered protocol.

ProtocolSpecificInfo
   This field contains the protocol-specified information that may
   be useful for a protocol to apply the key correctly. Note that
   such information MUST NOT be required for a protocol to locate
   an appropriate key. When a protocol does not need the
   information in ProtocolSpecificInfo, it will require this field
   be empty. Key table rows MAY specify a Direction of "both".
   As a result, the encoding of this field needs to support
   encoding protocol specific information for sending and
   receiving in the same row.

KDF

   The KDF field indicates the key derivation function which is
   used to generate short-lived keys from the long-lived value in
   the Key field. When the long-lived value in the Key field is
   intended for direct use, the KDF field is set to "none". A key
   derivation function is a one-way function that provides
   cryptographic separation of key material. The KDF MAY use
   inputs from the row in the key table and the message being sent
   or received but MUST NOT depend on other configuration state.
   This document establishes an IANA registry for the values in
   the KDF field to simplify references in future specifications.
   The protocol indicates what (if any) KDFs are valid.

AlgID
   The AlgID field indicates which cryptographic algorithm to be
   used with the security protocol for the specified peer or
   peers. Such an algorithm can be an encryption algorithm and
   mode (e.g., AES-128-CBC), an authentication algorithm (e.g.,
   HMAC-SHA1-96 or AES-128-CMAC), or any other symmetric
   cryptographic algorithm needed by a security protocol. If the
   KDF field contains "none", then the long-lived key is used
   directly with this algorithm, otherwise the derived short-lived
   key is used with this algorithm. When the long-lived key is
   used to generate a set of short-lived keys for use with the
   security protocol, the AlgID field identifies a ciphersuite
   rather than a single cryptographic algorithm. This document
   establishes an IANA registry for the values in the AlgID field
   to simplify references in future specifications. Protocols
   indicate which algorithms are appropriate.

Key
   The Key field contains a long-lived symmetric cryptographic key
   in the format of a lower-case hexadecimal string. The size of
   the Key depends on the KDF and the AlgID. For instance, a
   KDF=none and AlgID=AES128 requires a 128-bit key, which is
   represented by 32 hexadecimal digits.

Direction
    The Direction field indicates whether this key may be used for
    inbound traffic, outbound traffic, both, or whether the key
    has been disabled and may not currently be used at all. The
    supported values are "in", "out", "both", and "disabled",
    respectively. The Protocol field will determine which of these
    values are valid.

SendLifetimeStart
    The SendLifetimeStart field specifies the earliest date and
    time in Coordinated Universal Time (UTC) at which this key
    should be considered for use when sending traffic. The format
    is YYYYMMDDHHSSZ, where four digits specify the year, two
    digits specify the month, two digits specify the day, two
    digits specify the hour, two digits specify the minute, and
    two digits specify the second. The "Z" is included as a clear
    indication that the time is in UTC.

SendLifeTimeEnd
    The SendLifeTimeEnd field specifies the latest date and time at
    which this key should be considered for use when sending
    traffic. The format is the same as the SendLifetimeStart
    field.

AcceptLifeTimeStart
    The AcceptLifeTimeStart field specifies the earliest date and
    time in Coordinated Universal Time (UTC) at which this key
    should be considered for use when processing received traffic.
    The format is YYYYMMDDHHSSZ, where four digits specify the
    year, two digits specify the month, two digits specify the day,
    two digits specify the hour, two digits specify the minute, and
    two digits specify the second. The "Z" is included as a clear
    indication that the time is in UTC.

AcceptLifeTimeEnd
    The AcceptLifeTimeEnd field specifies the latest date and time
    at which this key should be considered for use when processing
    the received traffic. The format of this field is identical to
    the format of AcceptLifeTimeStart.

3. Key Selection and Rollover

A protocol may directly consult the key table to find the key to use
on an outgoing message. The protocol provides a protocol (P) and a
peer identifier (H) into the key selection function. Optionally, an
interface identifier (I) may also need to be provided. Any key that

satisfies the following conditions may be selected:

    (1) the Peers field includes H;

    (2) the Protocol field matches P;

    (3) If an interface is specified by the protocol, the Interfaces
    field in the key table row includes I or "all";

    (4) the Direction field is either "out" or "both"; and

    (5) SendLifetimeStart <= current time <= SendLifeTimeEnd.

During key selection, multiple entries may simultaneously exist
associated with different cryptographic algorithms or ciphersuites.
Systems should support selection of keys based on algorithm
preference to facilitate algorithm transition.

In addition, multiple entries with overlapping valid periods are
expected to be available for orderly key rollover. In these cases,
the expectation is that systems will transition to the newest key
available. To meet this requirement, this specification recommends
supplementing the key selection algorithm with the following
differentiation: select the long-lived key specifying the most recent
time in the SendLifetimeStart field.

In order to look up a key for validating an incoming message, the
protocol provides its protocol (P), the peer identifier (H), the key
identifier (L), and optionally the interface (I). If one key matches
the following conditions it is selected:

    (1) the Peer field includes H;

    (2) the Protocol field matches P;

    (3) if the Interface field is provided, it includes I or is
    "all";

    (4) the Direction field is either "in" or "both";

    (5) the LocalKeyName is L; and

    (6) AcceptLifeTimeStart <= current time <= AcceptLifeTimeEnd.

Note that the key usage is loosely bound by the times specified in
the AcceptLifeTimeStart and AcceptLifeTimeEnd fields. New security
associations should not be established except within the period of
use specified by these fields, while allowing some grace time for

   clock skew. However, if a security association has already been
   established based on a particular long-lived key, exceeding the
   lifetime does not have any direct impact. The implementations of
   security protocols that involve long-lived security association
   should be designed to periodically interrogate the database and
   rollover to new keys without tearing down the security association.

   Rather than consulting the conceptual database, a security protocol
   such as TCP-AO may update its own tables as keys are added and
   removed. In this case, the protocol needs to maintain its own key
   information. Some routing protocols use IP Security (IPsec) to
   provide integrity. If a specification describes how to use the
   conceptual database described in this document to configure keys for
   these routing protocols, similar concerns apply. The specification
   mapping those routing protocols onto this conceptual database needs
   to describe how the Security Policy Database is manipulated as rows
   are added and removed from the conceptual database.

4. Application of the Database in a Security Protocol

   In order to use the key table database in a protocol specification, a
   protocol needs to specify certain information. This section
   enumerates items that a protocol must specify.

      (1) The ways of mapping the information in a key table row to the
      information needed to produce an outgoing message; specified
      either as an explanation of how to fill in authentication-related
      fields in a message based on key table information, or for
      protocols such as TCP-AO how to construct Master Key Tuples (MKTs)
      or other protocol-specific structures from a key table row

      (2) The ways of locating the peer identifier (a member of the
      Peers set) and the LocalKeyName inside an incoming message

      (3) The methods of verifying a message given a key table row;
      this may be stated directly or in terms of protocol-specific
      structures such as MKTs

      (4) The form and validation rules for LocalKeyName and
      PeerKeyName; if either of these is an integer, the conventions in
      Section 5.1 are used as a vendor-independent format

      (5) The form and validation rules for members of the Peers set

      (6) The algorithms and KDFs supported

      (7) The form of the ProtocolSpecifics field

   (8) The rules for canonicalizing LocalKeyName, PeerKeyName,
   entries in the Peers set, or ProtocolSpecifics; this may include
   normalizations such as lower-casing hexadecimal strings

   (9) The Indication whether the support for Interfaces is required
   by this protocol

The form of the interfaces field is not protocol-specific but instead
is shared among all protocols on an implementation. If a protocol
needs to distinguish instances running over the same interface, this
is included in the specification of peers. Generally it is desirable
to define the specification of peers so that an operator can use the
interfaces field to refer to all instances of a protocol on a link
without having to specify both generic interfaces information and
protocol-specific peer information.

5. Textual Conventions

5.1 Key Names

When a key for a given protocol is identified by an integer key
identifier, the associated key name will be represented as lower case
hexadecimal digits with the most significant octet first. This
integer is padded with leading zero digits until the width of the key
identifier field in the protocol is reached. If a key name contains
non-integer human-readable text, its format and encoding may be an
issue, particularly if it is used in protocol between two different
types of systems. If characters from the ASCII range [RFC20] are
sufficient for a key name, then they SHOULD be used. If characters
outside of that range are desirable or required, then they MUST be in
an encoding of Unicode [UNICODE].

In the case of an AdminKeyName that uses characters outside of the
ASCII range, the AdminKeyName MUST be encoded using UTF-8 [RFC3629]
and SHOULD be normalized using Unicode Normalization Form KC [UAX15]
to maximize the chance that the strings will compare correctly.
However, simply using Unicode Normalization Form KC is not sufficient
to account for all issues of string comparison; refer to [draft-ietf-
precis-framework] for additional information.

5.2 Keys

A key is represented as a lower-case hexadecimal string with the most
significant octet of the key first. As discussed in Section 2, the
length of this string depends on the associated algorithm and KDF.

6. Operational Considerations

   If the valid periods for long-lived keys do not overlap or the system
   clocks are inconsistent, it is possible to construct scenarios where
   systems cannot agree upon a long-lived key. When installing a series
   of keys to be used one after another, operators should configure the
   SendLifetimeStart field of the key to be several hours after the
   AcceptLifeTimeStart field of the key to guarantee there is some
   overlap. This overlap is intended to address the clock skew issue and
   allow for basic operational considerations. Operators may choose to
   specify a longer overlap (e.g., several days) to allow for
   exceptional circumstances.

7. Security Considerations

   Management of encryption and authentication keys has been a
   significant operational problem, both in terms of key synchronization
   and key selection. For instance, the current guidance [RFC3562]
   warns against sharing TCP MD5 keying material between systems, and
   recommends changing keys according to a schedule. The same general
   operational issues are relevant for the management of other
   cryptographic keys.

   It has been recognized in [RFC4107] that automated key management is
   not viable in multiple scenarios. The conceptual database specified
   in this document is designed to accommodate both manual key
   management and automated key management. A future specification to
   automatically populate rows in the database is envisioned.

   Designers should recognize the warning provided in [RFC4107]:

      Automated key management and manual key management provide very
      different features. In particular, the protocol associated with
      an automated key management technique will confirm the liveness of
      the peer, protect against replay, authenticate the source of the
      short-term session key, associate protocol state information with
      the short-term session key, and ensure that a fresh short-term
      session key is generated. Moreover, an automated key management
      protocol can improve the interoperability by including negotiation
      mechanisms for cryptographic algorithms. These valuable features
      are impossible or extremely cumbersome to accomplish with manual
      key management.

8. IANA Considerations

   This specification defines three registries.

8.1. KeyTable Protocols

This document requests establishment of a registry called "KeyTable Protocols".

All assignments to the KeyTable Protocols registry are made on a specification required basis per Section 4.1 of [RFC5226].

Each registration entry must contain the three fields:

   - Protocol Name (unique within the registry);
   - Protocol Specific Info; and
   - Specification.

The specification needs to describe parameters required for using the conceptual database as outlined in Section 4. This typically means that the specification focuses more on the application of security protocols with the key tables rather than being a new security protocol specification for general purposes. New protocols may of course combine information on how to use the key tables database with the protocol specification.

The registry has three columns. The first column is a string of UTF-8 characters representing the name protocol. The second column is a string of UTF-8 characters providing a brief description of Protocol Specific Info. The third column is a reference to a specification defining how the protocol is used with the key table.

There are no initial registrations.

8.2. KeyTable KDFs

This document requests the establishment of a registry called "KeyTable KDFs".

All assignments to the KeyTable KDFs registry are made on a First Come First Served basis per Section 4.1 of RFC 5226.

The registry has three columns. The first column is a string of UTF-8 characters representing the name of a KDF. The second column is a string of UTF-8 characters providing a brief description of the KDF. The third column is a reference to a specification defining the KDF, if available.

```
   KDF             Description                   Reference
   ------------    -----------------------       ---------
   none            No KDF is used with this key  N/A
   AES-128-CMAC    AES-CMAC using 128-bit keys   [RFC4493]
   HMAC-SHA-1      HMAC using the SHA-1 hash      [RFC 2104]
```

8.3. KeyTable AlgIDs

This document requests establishment of a registry called "KeyTable
AlgIDs".

All assignments to the KeyTable AlgIDs registry are made on a First
Come First Served basis per Section 4.1 of RFC 5226.

The registry has three columns. The first column is a string of
UTF-8 characters representing the algorithm identifier (AlgID). The
second column is a string of UTF-8 characters providing a brief
description of the identified algorithm. The third column is a
reference to a specification defining the identified algorithm.

```
   AlgID             Description                      Reference
   ------------      ------------------------------   ---------
   AES-128-CMAC      AES-CMAC using 128-bit keys      [RFC4493]
   AES-128-CMAC-96   AES-128-CMAc truncated to 96-bits  [RFC5926]
   HMAC-SHA-1-96     HMAC SHA-1 truncated to 96-bits  [RFC2104]
```

9. Acknowledgments

This document reflects many discussions with many different people
over many years. In particular, the authors thank Jari Arkko, Ran
Atkinson, Ron Bonica, Ross Callon, Lars Eggert, Pasi Eronen, Adrian
Farrel, Gregory Lebovitz, Acee Lindem, Sandy Murphy, Eric Rescorla,
Mike Shand, Dave Ward, and Brian Weis for their insights. The
authors additionally thank Brian Weis for supplying text to address
IANA concerns and for help with formatting.

Sam Hartman's work on this draft is funded by Huawei.

10. References

10.1. Normative References

   [RFC20]   Cerf, V., "ASCII format for network interchange", RFC 20,
               October 1969.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
                Requirement Levels", BCP 14, RFC 2119, March 1997.

   [UAX15]   The Unicode Consortium, "Unicode Standard Annex #15:
              Unicode Normalization Forms", August 2012,
              <http://unicode.org/reports/tr15/>.

   [UNICODE] The Unicode Consortium, "The Unicode Standard", 2013,
              <http://www.unicode.org/versions/latest/>.

10.2. Informational References

   [draft-ietf-precis-framework]
              Saint-Andre, P. and M. Blanchet, "PRECIS Framework:
              Preparation and Comparison of Internationalized Strings
              in Application Protocols", work-in-progress,
              November 21, 2013.

   [IEEE802.1X-2010]
              IEEE Standard for Local and Metropolitan Area Networks --
              Port-Based Network Access Control", February 2010.

   [RFC3562] Leech, M., "Key Management Considerations for the TCP MD5
              Signature Option", RFC 3562, July 2003.

   [RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic
              Key Management", RFC 4107, BCP 107, June 2005.

   [RFC4493] Song, J., Lee, J., Poovendran, R., and T. Iwata, "The AES-CMAC
              Algorithm", RFC 4493, June 2006.

   [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
              IANA Considerations Section in RFCs", BCP 26, RFC 5226,
              May 2008.

   [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP
              Authentication Option", RFC 5925, June 2010.

Authors' Addresses

    Russell Housley
    Vigil Security, LLC
    918 Spring Knoll Drive
    Herndon, VA 20170
    USA
    EMail: housley@vigilsec.com

    Tim Polk
    National Institute of Standards and Technology
    100 Bureau Drive, Mail Stop 8930
    Gaithersburg, MD 20899-8930
    USA
    EMail: tim.polk@nist.gov

    Sam Hartman
    Painless Security, LLC
    USA
    Email: hartmans@painless-security.com

    Dacheng Zhang
    Huawei
    China
    Email: zhangdacheng@huawei.com