       Datagram Transport Layer Security (DTLS) Extension to Establish Keys
              for the Secure Real-time Transport Protocol (SRTP)

Abstract

   This document describes a Datagram Transport Layer Security (DTLS)
   extension to establish keys for Secure RTP (SRTP) and Secure RTP
   Control Protocol (SRTCP) flows. DTLS keying happens on the media
   path, independent of any out-of-band signalling channel present.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc5764.

Table of Contents

1. Introduction

   The Secure RTP (SRTP) profile [RFC3711] can provide confidentiality,
   message authentication, and replay protection to RTP data and RTP
   Control (RTCP) traffic. SRTP does not provide key management
   functionality, but instead depends on external key management to
   exchange secret master keys, and to negotiate the algorithms and
   parameters for use with those keys.

   Datagram Transport Layer Security (DTLS) [RFC4347] is a channel
   security protocol that offers integrated key management, parameter
   negotiation, and secure data transfer. Because DTLS data transfer
   protocol is generic, it is less highly optimized for use with RTP
   than is SRTP, which has been specifically tuned for that purpose.

   This document describes DTLS-SRTP, a SRTP extension for DTLS that
   combines the performance and encryption flexibility benefits of SRTP
   with the flexibility and convenience of DTLS-integrated key and
   association management. DTLS-SRTP can be viewed in two equivalent
   ways: as a new key management method for SRTP, and a new RTP-specific
   data format for DTLS.

   The key points of DTLS-SRTP are that:

   o application data is protected using SRTP,

   o the DTLS handshake is used to establish keying material,
      algorithms, and parameters for SRTP,

   o a DTLS extension is used to negotiate SRTP algorithms, and

   o other DTLS record-layer content types are protected using the
      ordinary DTLS record format.

   The remainder of this memo is structured as follows. Section 2
   describes conventions used to indicate normative requirements.
   Section 3 provides an overview of DTLS-SRTP operation. Section 4
   specifies the DTLS extensions, while Section 5 discusses how RTP and
   RTCP are transported over a DTLS-SRTP channel. Section 6 describes
   use with multi-party sessions. Section 7 and Section 9 describe
   Security and IANA considerations.

2. Conventions Used In This Document

   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].

3. Overview of DTLS-SRTP Operation

   DTLS-SRTP is defined for point-to-point media sessions, in which
   there are exactly two participants. Each DTLS-SRTP session contains
   a single DTLS association (called a "connection" in TLS jargon), and
   either two SRTP contexts (if media traffic is flowing in both
   directions on the same host/port quartet) or one SRTP context (if
   media traffic is only flowing in one direction). All SRTP traffic
   flowing over that pair in a given direction uses a single SRTP
   context. A single DTLS-SRTP session only protects data carried over
   a single UDP source and destination port pair.

   The general pattern of DTLS-SRTP is as follows. For each RTP or RTCP
   flow the peers do a DTLS handshake on the same source and destination
   port pair to establish a DTLS association. Which side is the DTLS
   client and which side is the DTLS server must be established via some
   out-of-band mechanism such as SDP. The keying material from that
   handshake is fed into the SRTP stack. Once that association is
   established, RTP packets are protected (becoming SRTP) using that
   keying material.

   RTP and RTCP traffic is usually sent on two separate UDP ports. When
   symmetric RTP [RFC4961] is used, two bidirectional DTLS-SRTP sessions
   are needed, one for the RTP port, one for the RTCP port. When RTP
   flows are not symmetric, four unidirectional DTLS-SRTP sessions are
   needed (for inbound and outbound RTP, and inbound and outbound RTCP).

   Symmetric RTP [RFC4961] is the case in which there are two RTP
   sessions that have their source and destination ports and addresses
   reversed, in a manner similar to the way that a TCP connection uses
   its ports. Each participant has an inbound RTP session and an
   outbound RTP session. When symmetric RTP is used, a single DTLS-SRTP
   session can protect both of the RTP sessions. It is RECOMMENDED that
   symmetric RTP be used with DTLS-SRTP.

   RTP and RTCP traffic MAY be multiplexed on a single UDP port
   [RFC5761]. In this case, both RTP and RTCP packets may be sent over
   the same DTLS-SRTP session, halving the number of DTLS-SRTP sessions
   needed. This improves the cryptographic performance of DTLS, but may
   cause problems when RTCP and RTP are subject to different network
   treatment (e.g., for bandwidth reservation or scheduling reasons).

   Between a single pair of participants, there may be multiple media
   sessions. There MUST be a separate DTLS-SRTP session for each
   distinct pair of source and destination ports used by a media session
   (though the sessions can share a single DTLS session and hence
   amortize the initial public key handshake!).

A DTLS-SRTP session may be indicated by an external signaling
protocol like SIP. When the signaling exchange is integrity-
protected (e.g., when SIP Identity protection via digital signatures
is used), DTLS-SRTP can leverage this integrity guarantee to provide
complete security of the media stream. A description of how to
indicate DTLS-SRTP sessions in SIP and SDP [RFC4566], and how to
authenticate the endpoints using fingerprints can be found in
[RFC5763].

In a naive implementation, when there are multiple media sessions,
there is a new DTLS session establishment (complete with public key
cryptography) for each media channel. For example, a videophone may
be sending both an audio stream and a video stream, each of which
would use a separate DTLS session establishment exchange, which would
proceed in parallel. As an optimization, the DTLS-SRTP
implementation SHOULD use the following strategy: a single DTLS
association is established, and all other DTLS associations wait
until that connection is established before proceeding with their
handshakes. This strategy allows the later sessions to use DTLS
session resumption, which allows the amortization of the expensive
public key cryptography operations over multiple DTLS handshakes.

The SRTP keys used to protect packets originated by the client are
distinct from the SRTP keys used to protect packets originated by the
server. All of the RTP sources originating on the client for the
same channel use the same SRTP keys, and similarly, all of the RTP
sources originating on the server for the same channel use the same
SRTP keys. The SRTP implementation MUST ensure that all of the
synchronization source (SSRC) values for all of the RTP sources
originating from the same device over the same channel are distinct,
in order to avoid the "two-time pad" problem (as described in Section
9.1 of RFC 3711). Note that this is not an issue for separate media
streams (on different host/port quartets) that use independent keying
material even if an SSRC collision occurs.

4. DTLS Extensions for SRTP Key Establishment

4.1. The use_srtp Extension

In order to negotiate the use of SRTP data protection, clients
include an extension of type "use_srtp" in the DTLS extended client
hello. This extension MUST only be used when the data being
transported is RTP or RTCP [RFC3550]. The "extension_data" field of
this extension contains the list of acceptable SRTP protection
profiles, as indicated below.

Servers that receive an extended hello containing a "use_srtp"
extension can agree to use SRTP by including an extension of type
"use_srtp", with the chosen protection profile in the extended server
hello. This process is shown below.

```
      Client                                               Server

      ClientHello + use_srtp        -------->
                                             ServerHello + use_srtp
                                                       Certificate*
                                                 ServerKeyExchange*
                                                CertificateRequest*
                                    <--------      ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      [ChangeCipherSpec]
      Finished                      -------->
                                                 [ChangeCipherSpec]
                                    <--------              Finished
      SRTP packets                  <------->          SRTP packets
```

Note that '*' indicates messages that are not always sent in DTLS.
The CertificateRequest, client and server Certificates, and
CertificateVerify will be sent in DTLS-SRTP.

Once the "use_srtp" extension is negotiated, the RTP or RTCP
application data is protected solely using SRTP. Application data is
never sent in DTLS record-layer "application_data" packets. Rather,
complete RTP or RTCP packets are passed to the DTLS stack, which
passes them to the SRTP stack, which protects them appropriately.
Note that if RTP/RTCP multiplexing [RFC5761] is in use, this means
that RTP and RTCP packets may both be passed to the DTLS stack.
Because the DTLS layer does not process the packets, it does not need
to distinguish them. The SRTP stack can use the procedures of
[RFC5761] to distinguish RTP from RTCP.

When the "use_srtp" extension is in effect, implementations must not
place more than one application data "record" per datagram. (This is
only meaningful from the perspective of DTLS because SRTP is
inherently oriented towards one payload per packet, but this is
stated purely for clarification.)

Data other than RTP/RTCP (i.e., TLS control messages) MUST use
ordinary DTLS framing and MUST be placed in separate datagrams from
SRTP data.

A DTLS-SRTP handshake establishes one or more SRTP crypto contexts;
however, they all have the same SRTP Protection Profile and Master
Key Identifier (MKI), if any. MKIs are used solely to distinguish
the keying material and protection profiles between distinct
handshakes, for instance, due to rekeying. When an MKI is
established in a DTLS-SRTP session, it MUST apply for all of the
SSRCs within that session -- though a single endpoint may negotiate
multiple DTLS-SRTP sessions due, for instance, to forking. (Note
that RFC 3711 allows packets within the same session but with
different SSRCs to use MKIs differently; in contrast, DTLS-SRTP
requires that MKIs and the keys that they are associated with have
the same meaning and are uniform across the entire SRTP session.)

4.1.1. use_srtp Extension Definition

The client MUST fill the extension_data field of the "use_srtp"
extension with an UseSRTPData value (see Section 9 for the
registration):

```
uint8 SRTPProtectionProfile[2];

struct {
   SRTPProtectionProfiles SRTPProtectionProfiles;
   opaque srtp_mki<0..255>;
} UseSRTPData;

SRTPProtectionProfile SRTPProtectionProfiles<2..2^16-1>;
```

The SRTPProtectionProfiles list indicates the SRTP protection
profiles that the client is willing to support, listed in descending
order of preference. The srtp_mki value contains the SRTP Master Key
Identifier (MKI) value (if any) that the client will use for his SRTP
packets. If this field is of zero length, then no MKI will be used.

Note: for those unfamiliar with TLS syntax, "srtp_mki<0..255>"
indicates a variable-length value with a length between 0 and 255
(inclusive). Thus, the MKI may be up to 255 bytes long.

If the server is willing to accept the use_srtp extension, it MUST
respond with its own "use_srtp" extension in the ExtendedServerHello.
The extension_data field MUST contain a UseSRTPData value with a
single SRTPProtectionProfile value that the server has chosen for use
with this connection. The server MUST NOT select a value that the
client has not offered. If there is no shared profile, the server
SHOULD NOT return the use_srtp extension at which point the
connection falls back to the negotiated DTLS cipher suite. If that
is not acceptable, the server SHOULD return an appropriate DTLS
alert.

4.1.2. SRTP Protection Profiles

   A DTLS-SRTP SRTP Protection Profile defines the parameters and
   options that are in effect for the SRTP processing. This document
   defines the following SRTP protection profiles.

      SRTPProtectionProfile SRTP_AES128_CM_HMAC_SHA1_80 = {0x00, 0x01};
      SRTPProtectionProfile SRTP_AES128_CM_HMAC_SHA1_32 = {0x00, 0x02};
      SRTPProtectionProfile SRTP_NULL_HMAC_SHA1_80   = {0x00, 0x05};
      SRTPProtectionProfile SRTP_NULL_HMAC_SHA1_32   = {0x00, 0x06};

   The following list indicates the SRTP transform parameters for each
   protection profile. The parameters cipher_key_length,
   cipher_salt_length, auth_key_length, and auth_tag_length express the
   number of bits in the values to which they refer. The
   maximum_lifetime parameter indicates the maximum number of packets
   that can be protected with each single set of keys when the parameter
   profile is in use. All of these parameters apply to both RTP and
   RTCP, unless the RTCP parameters are separately specified.

   All of the crypto algorithms in these profiles are from [RFC3711].

   SRTP_AES128_CM_HMAC_SHA1_80
         cipher: AES_128_CM
         cipher_key_length: 128
         cipher_salt_length: 112
         maximum_lifetime: 2^{31}
         auth_function: HMAC-SHA1
         auth_key_length: 160
         auth_tag_length: 80
   SRTP_AES128_CM_HMAC_SHA1_32
         cipher: AES_128_CM
         cipher_key_length: 128
         cipher_salt_length: 112
         maximum_lifetime: 2^{31}
         auth_function: HMAC-SHA1
         auth_key_length: 160
         auth_tag_length: 32
         RTCP auth_tag_length: 80
   SRTP_NULL_HMAC_SHA1_80
         cipher: NULL
         cipher_key_length: 0
         cipher_salt_length: 0
         maximum_lifetime: 2^{31}
         auth_function: HMAC-SHA1
         auth_key_length: 160
         auth_tag_length: 80

```
SRTP_NULL_HMAC_SHA1_32
      cipher: NULL
      cipher_key_length: 0
      cipher_salt_length: 0
      maximum_lifetime: 2^31
      auth_function: HMAC-SHA1
      auth_key_length: 160
      auth_tag_length: 32
      RTCP auth_tag_length: 80
```

With all of these SRTP Parameter profiles, the following SRTP options
are in effect:

o The TLS PseudoRandom Function (PRF) is used to generate keys to
   feed into the SRTP Key Derivation Function (KDF). When DTLS 1.2
   [DTLS1.2] is in use, the PRF is the one associated with the cipher
   suite. Note that this specification is compatible with DTLS 1.0
   or DTLS 1.2

o The Key Derivation Rate (KDR) is equal to zero. Thus, keys are
   not re-derived based on the SRTP sequence number.

o The key derivation procedures from Section 4.3 with the AES-CM PRF
   from RFC 3711 are used.

o For all other parameters (in particular, SRTP replay window size
   and FEC order), the default values are used.

If values other than the defaults for these parameters are required,
they can be enabled by writing a separate specification specifying
SDP syntax to signal them.

Applications using DTLS-SRTP SHOULD coordinate the SRTP Protection
Profiles between the DTLS-SRTP session that protects an RTP flow and
the DTLS-SRTP session that protects the associated RTCP flow (in
those cases in which the RTP and RTCP are not multiplexed over a
common port). In particular, identical ciphers SHOULD be used.

New SRTPProtectionProfile values must be defined according to the
"Specification Required" policy as defined by RFC 5226 [RFC5226].
See Section 9 for IANA Considerations.

4.1.3. srtp_mki value

The srtp_mki value MAY be used to indicate the capability and desire
to use the SRTP Master Key Identifier (MKI) field in the SRTP and
SRTCP packets. The MKI field indicates to an SRTP receiver which key
was used to protect the packet that contains that field. The

srtp_mki field contains the value of the SRTP MKI which is associated
with the SRTP master keys derived from this handshake. Each SRTP
session MUST have exactly one master key that is used to protect
packets at any given time. The client MUST choose the MKI value so
that it is distinct from the last MKI value that was used, and it
SHOULD make these values unique for the duration of the TLS session.

Upon receipt of a "use_srtp" extension containing a "srtp_mki" field,
the server MUST either (assuming it accepts the extension at all):

1. include a matching "srtp_mki" value in its "use_srtp" extension
    to indicate that it will make use of the MKI, or
2. return an empty "srtp_mki" value to indicate that it cannot make
    use of the MKI.

If the client detects a nonzero-length MKI in the server's response
that is different than the one the client offered, then the client
MUST abort the handshake and SHOULD send an invalid_parameter alert.
If the client and server agree on an MKI, all SRTP packets protected
under the new security parameters MUST contain that MKI.

Note that any given DTLS-SRTP session only has a single active MKI
(if any). Thus, at any given time, a set of endpoints will generally
only be using one MKI (the major exception is during rehandshakes).

4.2. Key Derivation

When SRTP mode is in effect, different keys are used for ordinary
DTLS record protection and SRTP packet protection. These keys are
generated using a TLS exporter [RFC5705] to generate

2 * (SRTPSecurityParams.master_key_len +
     SRTPSecurityParams.master_salt_len) bytes of data

which are assigned as shown below. The per-association context value
is empty.

client_write_SRTP_master_key[SRTPSecurityParams.master_key_len];
server_write_SRTP_master_key[SRTPSecurityParams.master_key_len];
client_write_SRTP_master_salt[SRTPSecurityParams.master_salt_len];
server_write_SRTP_master_salt[SRTPSecurityParams.master_salt_len];

The exporter label for this usage is "EXTRACTOR-dtls_srtp". (The
"EXTRACTOR" prefix is for historical compatibility.)

The four keying material values (the master key and master salt for
each direction) are provided as inputs to the SRTP key derivation
mechanism, as shown in Figure 1 and detailed below. By default, the

mechanism defined in Section 4.3 of [RFC3711] is used, unless another
key derivation mechanism is specified as part of an SRTP Protection
Profile.

The client_write_SRTP_master_key and client_write_SRTP_master_salt
are provided to one invocation of the SRTP key derivation function,
to generate the SRTP keys used to encrypt and authenticate packets
sent by the client. The server MUST only use these keys to decrypt
and to check the authenticity of inbound packets.

The server_write_SRTP_master_key and server_write_SRTP_master_salt
are provided to one invocation of the SRTP key derivation function,
to generate the SRTP keys used to encrypt and authenticate packets
sent by the server. The client MUST only use these keys to decrypt
and to check the authenticity of inbound packets.

```
TLS master
  secret    label
    |         |
    v         v
+--------------+
| TLS extractor |
+--------------+
        |                                    +------+   SRTP
        +-> client_write_SRTP_master_key ----+--->| SRTP |-> client
        |                                    | +->| KDF  |   write
        |                                    | |  +------+   keys
        |                                    | |
        +-> server_write_SRTP_master_key --  | |  +------+   SRTCP
        |                                  \ \--->|SRTCP |-> client
        |                                   \  +->| KDF  |   write
        |                                    | |  +------+   keys
        +-> client_write_SRTP_master_salt ---|-+
        |                                    |
        |                                    |  +------+   SRTP
        |                                  +--->| SRTP |-> server
        +-> server_write_SRTP_master_salt -+-|--->| KDF  |   write
                                           | |  +------+   keys
                                           | |
                                           | |  +------+   SRTCP
                                           | +--->|SRTCP |-> server
                                           +----->| KDF  |   write
                                                +------+   keys
```
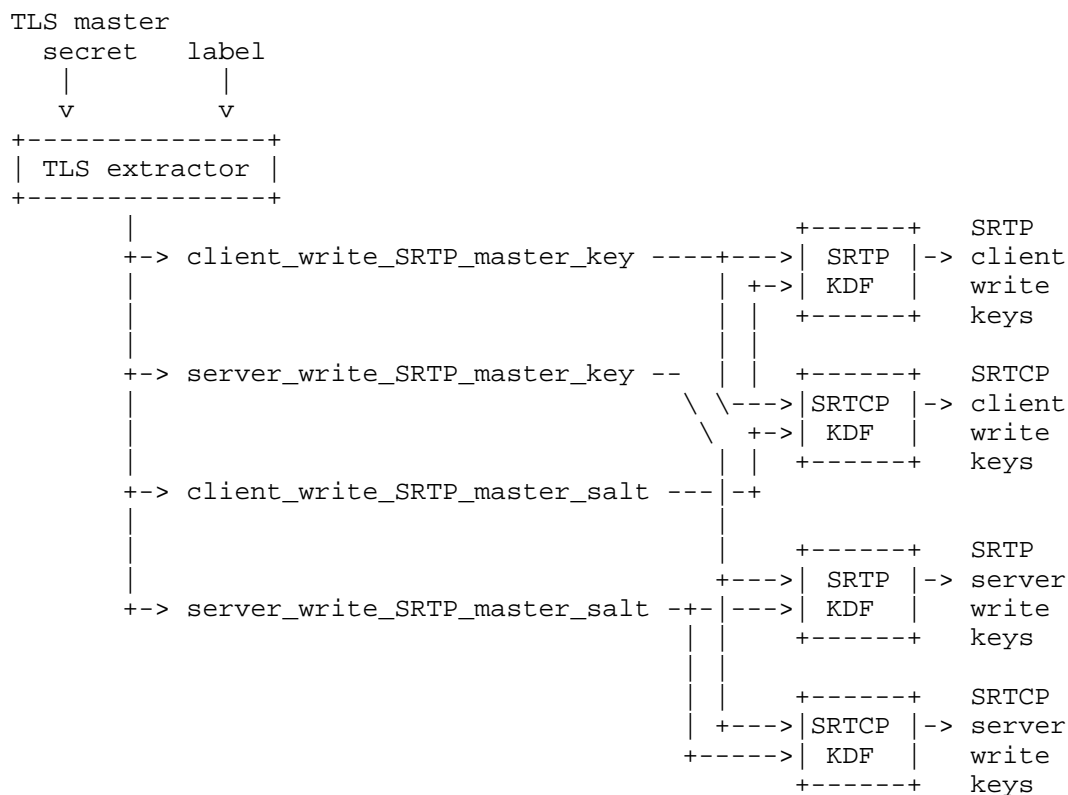
Figure 1: The derivation of the SRTP keys.

When both RTCP and RTP use the same source and destination ports,
then both the SRTP and SRTCP keys are needed. Otherwise, there are
two DTLS-SRTP sessions, one of which protects the RTP packets and one
of which protects the RTCP packets; each DTLS-SRTP session protects
the part of an SRTP session that passes over a single source/
destination transport address pair, as shown in Figure 2, independent
of which SSRCs are used on that pair. When a DTLS-SRTP session is
protecting RTP, the SRTCP keys derived from the DTLS handshake are
not needed and are discarded. When a DTLS-SRTP session is protecting
RTCP, the SRTP keys derived from the DTLS handshake are not needed
and are discarded.

```
     Client              Server
    (Sender)           (Receiver)
(1)    <----- DTLS ------>    src/dst = a/b and b/a
       ------ SRTP ------>    src/dst = a/b, uses client write keys

(2)    <----- DTLS ------>    src/dst = c/d and d/c
       ------ SRTCP ----->    src/dst = c/d, uses client write keys
       <----- SRTCP ------    src/dst = d/c, uses server write keys
```

   Figure 2: A DTLS-SRTP session protecting RTP (1) and another one
  protecting RTCP (2), showing the transport addresses and keys used.

4.3. Key Scope

   Because of the possibility of packet reordering, DTLS-SRTP
   implementations SHOULD store multiple SRTP keys sets during a rekey
   in order to avoid the need for receivers to drop packets for which
   they lack a key.

4.4. Key Usage Limitations

   The maximum_lifetime parameter in the SRTP protection profile
   indicates the maximum number of packets that can be protected with
   each single encryption and authentication key. (Note that, since RTP
   and RTCP are protected with independent keys, those protocols are
   counted separately for the purposes of determining when a key has
   reached the end of its lifetime.) Each profile defines its own
   limit. When this limit is reached, a new DTLS session SHOULD be used
   to establish replacement keys, and SRTP implementations MUST NOT use
   the existing keys for the processing of either outbound or inbound
   traffic.

5. Use of RTP and RTCP over a DTLS-SRTP Channel

5.1. Data Protection

   Once the DTLS handshake has completed, the peers can send RTP or RTCP
   over the newly created channel. We describe the transmission process
   first followed by the reception process.

   Within each RTP session, SRTP processing MUST NOT take place before
   the DTLS handshake completes.

5.1.1. Transmission

   DTLS and TLS define a number of record content types. In ordinary
   TLS/DTLS, all data is protected using the same record encoding and
   mechanisms. When the mechanism described in this document is in
   effect, this is modified so that data written by upper-level protocol
   clients of DTLS is assumed to be RTP/RTP and is encrypted using SRTP
   rather than the standard TLS record encoding.

   When a user of DTLS wishes to send an RTP packet in SRTP mode, it
   delivers it to the DTLS implementation as an ordinary application
   data write (e.g., SSL_write()). The DTLS implementation then invokes
   the processing described in RFC 3711, Sections 3 and 4. The
   resulting SRTP packet is then sent directly on the wire as a single
   datagram with no DTLS framing. This provides an encapsulation of the
   data that conforms to and interoperates with SRTP. Note that the RTP
   sequence number rather than the DTLS sequence number is used for
   these packets.

5.1.2. Reception

   When DTLS-SRTP is used to protect an RTP session, the RTP receiver
   needs to demultiplex packets that are arriving on the RTP port.
   Arriving packets may be of types RTP, DTLS, or STUN [RFC5389]. If
   these are the only types of packets present, the type of a packet can
   be determined by looking at its first byte.

   The process for demultiplexing a packet is as follows. The receiver
   looks at the first byte of the packet. If the value of this byte is
   0 or 1, then the packet is STUN. If the value is in between 128 and
   191 (inclusive), then the packet is RTP (or RTCP, if both RTCP and
   RTP are being multiplexed over the same destination port). If the
   value is between 20 and 63 (inclusive), the packet is DTLS. This
   process is summarized in Figure 3.

```
              +---------------+
              | 127 < B < 192 -+--> forward to RTP
              |               |
   packet --> |  19 < B < 64  -+--> forward to DTLS
              |               |
              |      B < 2    -+--> forward to STUN
              +---------------+
```
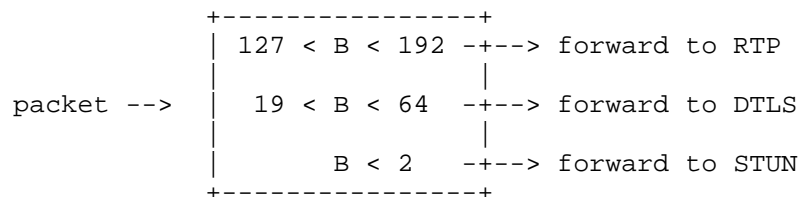
   Figure 3: The DTLS-SRTP receiver's packet demultiplexing algorithm.
        Here the field B denotes the leading byte of the packet.

   If other packet types are to be multiplexed as well, implementors
   and/or designers SHOULD ensure that they can be demultiplexed from
   these three packet types.

   In some cases, there will be multiple DTLS-SRTP associations for a
   given SRTP endpoint. For instance, if Alice makes a call that is SIP
   forked to both Bob and Charlie, she will use the same local host/port
   pair for both of them, as shown in Figure 4, where XXX and YYY
   represent different DTLS-SRTP associations. (The SSRCs shown are the
   ones for data flowing to Alice.)

```
                                    Bob (192.0.2.1:6666)
                                  /
                                /
                              / SSRC=1
                            /   DTLS-SRTP=XXX
                          /
                         v
            Alice (192.0.2.0:5555)
                          ^
                           \
                            \
                             \  SSRC=2
                              \ DTLS-SRTP=YYY
                               \
                                \
                                 Charlie (192.0.2.2:6666)
```
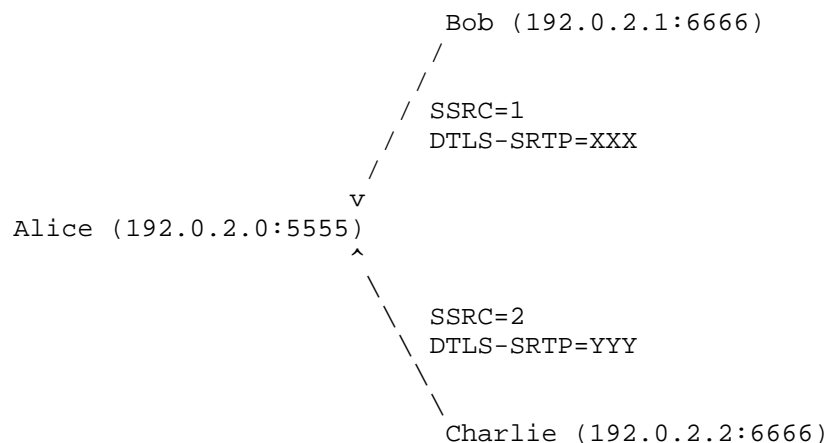
              Figure 4: RTP sessions with SIP forking.

   Because DTLS operates on the host/port quartet, the DTLS association
   will still complete correctly, with the foreign host/port pair being
   used, to distinguish the associations. However, in RTP the source
   host/port is not used and sessions are identified by the destination
   host/port and the SSRC. Thus, some mechanism is needed to determine
   which SSRCs correspond to which DTLS associations. The following
   method SHOULD be used.

For each local host/port pair, the DTLS-SRTP implementation maintains
a table listing all the SSRCs it knows about and the DTLS-SRTP
associations they correspond to. Initially, this table is empty.
When an SRTP packet is received for a given RTP endpoint (destination
IP/port pair), the following procedure is used:

1. If the SSRC is already known for that endpoint, then the
    corresponding DTLS-SRTP association and its keying material is
    used to decrypt and verify the packet.
2. If the SSRC is not known, then the receiver tries to decrypt it
    with the keying material corresponding to each DTLS-SRTP
    association for that endpoint.
3. If the decryption and verification succeeds (the authentication
    tag verifies), then an entry is placed in the table mapping the
    SSRC to that association.
4. If the decryption and verification fails, then the packet is
    silently discarded.
5. When a DTLS-SRTP association is closed (for instance, because the
    fork is abandoned), its entries MUST be removed from the mapping
    table.

The average cost of this algorithm for a single SSRC is the
decryption and verification time of a single packet times the number
of valid DTLS-SRTP associations corresponding to a single receiving
port on the host. In practice, this means the number of forks; so in
the case shown in Figure 4, that would be two. This cost is only
incurred once for any given SSRC, since afterwards that SSRC is
placed in the map table and looked up immediately. As with normal
RTP, this algorithm allows new SSRCs to be introduced by the source
at any time. They will automatically be mapped to the correct DTLS
association.

Note that this algorithm explicitly allows multiple SSRCs to be sent
from the same address/port pair. One way in which this can happen is
an RTP translator. This algorithm will automatically assign the
SSRCs to the correct associations. Note that because the SRTP
packets are cryptographically protected, such a translator must
either share keying material with one endpoint or refrain from
modifying the packets in a way which would cause the integrity check
to fail. This is a general property of SRTP and is not specific to
DTLS-SRTP.

There are two error cases that should be considered. First, if an
SSRC collision occurs, then only the packets from the first source
will be processed. When the packets from the second source arrive,
the DTLS association with the first source will be used for
decryption and verification, which will fail, and the packet will be
discarded. This is consistent with [RFC3550], which permits the

receiver to keep the packets from one source and discard those from
the other. Of course the RFC 3550 SSRC collision detection and
handling procedures MUST also be followed.

Second, there may be cases where a malfunctioning source is sending
corrupt packets that cannot be decrypted and verified. In this case,
the SSRC will never be entered into the mapping table because the
decryption and verification always fails. Receivers MAY keep records
of unmapped SSRCs that consistently fail decryption and verification
and abandon attempts to process them once they reach some limit.
That limit MUST be large enough to account for the effects of
transmission errors. Entries MUST be pruned from this table when the
relevant SRTP endpoint is deleted (e.g., the call ends) and SHOULD
time out faster than that (we do not offer a hard recommendation but
10 to 30 seconds seems appropriate) in order to allow for the
possibility that the peer implementation has been corrected.

5.2. Rehandshake and Rekey

Rekeying in DTLS is accomplished by performing a new handshake over
the existing DTLS channel. That is, the handshake messages are
protected by the existing DTLS cipher suite. This handshake can be
performed in parallel with data transport, so no interruption of the
data flow is required. Once the handshake is finished, the newly
derived set of keys is used to protect all outbound packets, both
DTLS and SRTP.

Because of packet reordering, packets protected by the previous set
of keys can appear on the wire after the handshake has completed. To
compensate for this fact, receivers SHOULD maintain both sets of keys
for some time in order to be able to decrypt and verify older
packets. The keys should be maintained for the duration of the
maximum segment lifetime (MSL).

If an MKI is used, then the receiver should use the corresponding set
of keys to process an incoming packet. If no matching MKI is
present, the packet MUST be rejected. Otherwise, when a packet
arrives after the handshake completed, a receiver SHOULD use the
newly derived set of keys to process that packet unless there is an
MKI. (If the packet was protected with the older set of keys, this
fact will become apparent to the receiver as an authentication
failure will occur.) If the authentication check on the packet fails
and no MKI is being used, then the receiver MAY process the packet
with the older set of keys. If that authentication check indicates
that the packet is valid, the packet should be accepted; otherwise,
the packet MUST be discarded and rejected.

   Receivers MAY use the SRTP packet sequence number to aid in the
   selection of keys. After a packet has been received and
   authenticated with the new key set, any packets with sequence numbers
   that are greater will also have been protected with the new key set.

6. Multi-Party RTP Sessions

   Since DTLS is a point-to-point protocol, DTLS-SRTP is intended only
   to protect unicast RTP sessions. This does not preclude its use with
   RTP mixers. For example, a conference bridge may use DTLS-SRTP to
   secure the communication to and from each of the participants in a
   conference. However, because each flow between an endpoint and a
   mixer has its own key, the mixer has to decrypt and then reencrypt
   the traffic for each recipient.

   A future specification may describe methods for sharing a single key
   between multiple DTLS-SRTP associations thus allowing conferencing
   systems to avoid the decrypt/reencrypt stage. However, any system in
   which the media is modified (e.g., for level balancing or
   transcoding) will generally need to be performed on the plaintext and
   will certainly break the authentication tag, and therefore will
   require a decrypt/reencrypt stage.

7. Security Considerations

   The use of multiple data protection framings negotiated in the same
   handshake creates some complexities, which are discussed here.

7.1. Security of Negotiation

   One concern here is that attackers might be able to implement a bid-
   down attack forcing the peers to use ordinary DTLS rather than SRTP.
   However, because the negotiation of this extension is performed in
   the DTLS handshake, it is protected by the Finished messages.
   Therefore, any bid-down attack is automatically detected, which
   reduces this to a denial-of-service attack -- which can be mounted by
   any attacker who can control the channel.

7.2. Framing Confusion

   Because two different framing formats are used, there is concern that
   an attacker could convince the receiver to treat an SRTP-framed RTP
   packet as a DTLS record (e.g., a handshake message) or vice versa.
   This attack is prevented by using different keys for Message
   Authentication Code (MAC) verification for each type of data.
   Therefore, this type of attack reduces to being able to forge a
   packet with a valid MAC, which violates a basic security invariant of
   both DTLS and SRTP.

As an additional defense against injection into the DTLS handshake
channel, the DTLS record type is included in the MAC. Therefore, an
SRTP record would be treated as an unknown type and ignored. (See
Section 6 of [RFC5246].)

7.3. Sequence Number Interactions

As described in Section 5.1.1, the SRTP and DTLS sequence number
spaces are distinct. This means that it is not possible to
unambiguously order a given DTLS control record with respect to an
SRTP packet. In general, this is relevant in two situations: alerts
and rehandshake.

7.3.1. Alerts

Because DTLS handshake and change_cipher_spec messages share the same
sequence number space as alerts, they can be ordered correctly.
Because DTLS alerts are inherently unreliable and SHOULD NOT be
generated as a response to data packets, reliable sequencing between
SRTP packets and DTLS alerts is not an important feature. However,
implementations that wish to use DTLS alerts to signal problems with
the SRTP encoding SHOULD simply act on alerts as soon as they are
received and assume that they refer to the temporally contiguous
stream. Such implementations MUST check for alert retransmission and
discard retransmitted alerts to avoid overreacting to replay attacks.

7.3.2. Renegotiation

Because the rehandshake transition algorithm specified in Section 5.2
requires trying multiple sets of keys if no MKI is used, it slightly
weakens the authentication. For instance, if an n-bit MAC is used
and k different sets of keys are present, then the MAC is weakened by
$\log\_2(k)$ bits to $n - \log\_2(k)$. In practice, since the number of keys
used will be very small and the MACs in use are typically strong (the
default for SRTP is 80 bits), the decrease in security involved here
is minimal.

Another concern here is that this algorithm slightly increases the
work factor on the receiver because it needs to attempt multiple
validations. However, again, the number of potential keys will be
very small (and the attacker cannot force it to be larger) and this
technique is already used for rollover counter management, so the
authors do not consider this to be a serious flaw.

7.4. Decryption Cost

   An attacker can impose computational costs on the receiver by sending
   superficially valid SRTP packets that do not decrypt correctly. In
   general, encryption algorithms are so fast that this cost is
   extremely small compared to the bandwidth consumed. The SSRC-DTLS
   mapping algorithm described in Section 5.1.2 gives the attacker a
   slight advantage here because he can force the receiver to do more
   then one decryption per packet. However, this advantage is modest
   because the number of decryptions that the receiver does is limited
   by the number of associations he has corresponding to a given
   destination host/port, which is typically quite small. For
   comparison, a single 1024-bit RSA private key operation (the typical
   minimum cost to establish a DTLS-SRTP association) is hundreds of
   times as expensive as decrypting an SRTP packet.

   Implementations can detect this form of attack by keeping track of
   the number of SRTP packets that are observed with unknown SSRCs and
   that fail the authentication tag check. If under such attack,
   implementations SHOULD prioritize decryption and verification of
   packets that either have known SSRCs or come from source addresses
   that match those of peers with which it has DTLS-SRTP associations.

8. Session Description for RTP/SAVP over DTLS

   This specification defines new tokens to describe the protocol used
   in SDP media descriptions ("m=" lines and their associated
   parameters). The new values defined for the proto field are:

   o When a RTP/SAVP or RTP/SAVPF [RFC5124] stream is transported over
      DTLS with the Datagram Congestion Control Protocol (DCCP), then
      the token SHALL be DCCP/TLS/RTP/SAVP or DCCP/TLS/RTP/SAVPF
      respectively.

   o When a RTP/SAVP or RTP/SAVPF stream is transported over DTLS with
      UDP, the token SHALL be UDP/TLS/RTP/SAVP or UDP/TLS/RTP/SAVPF
      respectively.

   The "fmt" parameter SHALL be as defined for RTP/SAVP.

   See [RFC5763] for how to use offer/answer with DTLS-SRTP.

   This document does not specify how to protect RTP data transported
   over TCP. Potential approaches include carrying the RTP over TLS
   over TCP (see [SRTP-NOT-MAND]) or using a mechanism similar to that
   in this document over TCP, either via TLS or DTLS, with DTLS being
   used for consistency between reliable and unreliable transports. In

the latter case, it would be necessary to profile DTLS so that
fragmentation and retransmissions no longer occurred. In either
case, a new document would be required.

9. IANA Considerations

   This document adds a new extension for DTLS, in accordance with
   [RFC5246]:
        enum { use_srtp (14) } ExtensionType;

   This extension MUST only be used with DTLS, and not with TLS
   [RFC4572], which specifies that TLS can be used over TCP but does not
   address TCP for RTP/SAVP.

   Section 4.1.2 requires that all SRTPProtectionProfile values be
   defined by RFC 5226 "Specification Required". IANA has created a
   DTLS SRTPProtectionProfile registry initially populated with values
   from Section 4.1.2 of this document. Future values MUST be allocated
   via the "Specification Required" profile of [RFC5226].

   This specification updates the "Session Description Protocol (SDP)
   Parameters" registry as defined in Section 8.2.2 of [RFC4566].
   Specifically, it adds the following values to the table for the
   "proto" field.

          Type   SDP Name   Reference
          ----   ------------------   ---------
          proto   UDP/TLS/RTP/SAVP   [RFC5764]
          proto   DCCP/TLS/RTP/SAVP   [RFC5764]

          proto   UDP/TLS/RTP/SAVPF   [RFC5764]
          proto   DCCP/TLS/RTP/SAVPF   [RFC5764]

   IANA has registered the "EXTRACTOR-dtls_srtp" value in the TLS
   Extractor Label Registry to correspond to this specification.

10. Acknowledgments

   Special thanks to Flemming Andreasen, Francois Audet, Pasi Eronen,
   Roni Even, Jason Fischl, Cullen Jennings, Colin Perkins, Dan Wing,
   and Ben Campbell for input, discussions, and guidance. Pasi Eronen
   provided Figure 1.

11. References

11.1. Normative References

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

[RFC3711]   Baugher, M., McGrew, D., Naslund, M., Carrara, E.,
                 and K. Norrman, "The Secure Real-time Transport
                 Protocol (SRTP)", RFC 3711, March 2004.

[RFC4347]   Rescorla, E. and N. Modadugu, "Datagram Transport
                 Layer Security", RFC 4347, April 2006.

[RFC4961]   Wing, D., "Symmetric RTP / RTP Control Protocol
                 (RTCP)", BCP 131, RFC 4961, July 2007.

[RFC5246]   Dierks, T. and E. Rescorla, "The Transport Layer
                 Security (TLS) Protocol Version 1.2", RFC 5246,
                 August 2008.

[RFC5705]   Rescorla, E., "Keying Material Exporters for
                 Transport Layer Security (TLS)", RFC 5705,
                 March 2010.

[RFC5761]   Perkins, C. and M. Westerlund, "Multiplexing RTP
                 Data and Control Packets on a Single Port",
                 RFC 5761, April 2010.

11.2. Informative References

[DTLS1.2]   Rescorla, E. and N. Modadugu, "Datagram Transport
                 Layer Security version 1.2", Work in Progress,
                 October 2009.

[RFC3550]   Schulzrinne, H., Casner, S., Frederick, R., and V.
                 Jacobson, "RTP: A Transport Protocol for Real-Time
                 Applications", STD 64, RFC 3550, July 2003.

[RFC4566]   Handley, M., Jacobson, V., and C. Perkins, "SDP:
                 Session Description Protocol", RFC 4566, July 2006.

[RFC4572]   Lennox, J., "Connection-Oriented Media Transport
                 over the Transport Layer Security (TLS) Protocol in
                 the Session Description Protocol (SDP)", RFC 4572,
                 July 2006.

   [RFC5124]   Ott, J. and E. Carrara, "Extended Secure RTP Profile
                        for Real-time Transport Control Protocol (RTCP)-
                        Based Feedback (RTP/SAVPF)", RFC 5124,
                        February 2008.

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

   [RFC5389]   Rosenberg, J., Mahy, R., Matthews, P., and D. Wing,
                        "Session Traversal Utilities for NAT (STUN)",
                        RFC 5389, October 2008.

   [RFC5763]   Fischl, J., Tschofenig, H., and E. Rescorla,
                        "Framework for Establishing a Secure Real-time
                        Transport Protocol (SRTP) Security Context Using
                        Datagram Transport Layer Security (DTLS)", RFC 5763,
                        May 2010.

   [SRTP-NOT-MAND]   Perkins, C. and M. Westerlund, "Why RTP Does Not
                        Mandate a Single Security Mechanism", Work in
                        Progress, January 2010.

Appendix A. Overview of DTLS

   This section provides a brief overview of Datagram TLS (DTLS) for
   those who are not familiar with it. DTLS is a channel security
   protocol based on the well-known Transport Layer Security (TLS)
   [RFC5246] protocol. Where TLS depends on a reliable transport
   channel (typically TCP), DTLS has been adapted to support unreliable
   transports such as UDP. Otherwise, DTLS is nearly identical to TLS
   and generally supports the same cryptographic mechanisms.

   Each DTLS association begins with a handshake exchange (shown below)
   during which the peers authenticate each other and negotiate
   algorithms, modes, and other parameters and establish shared keying
   material, as shown below. In order to support unreliable transport,
   each side maintains retransmission timers to provide reliable
   delivery of these messages. Once the handshake is completed,
   encrypted data may be sent.
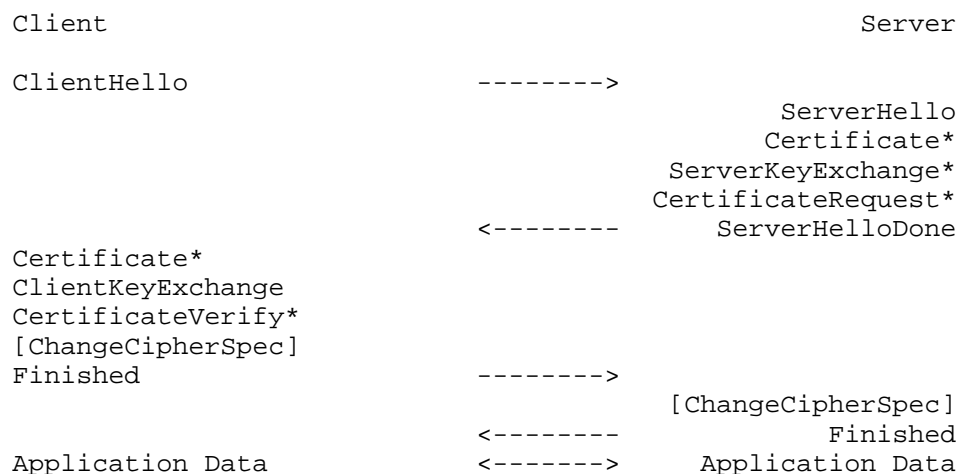
```
     Client                                              Server

     ClientHello                     -------->
                                                      ServerHello
                                                     Certificate*
                                               ServerKeyExchange*
                                              CertificateRequest*
                                     <--------      ServerHelloDone
     Certificate*
     ClientKeyExchange
     CertificateVerify*
     [ChangeCipherSpec]
     Finished                        -------->
                                                [ChangeCipherSpec]
                                     <--------             Finished
     Application Data                <------->     Application Data

             '*' indicates messages that are not always sent.
```

        Figure 5: Basic DTLS Handshake Exchange (after [RFC4347]).

   Application data is protected by being sent as a series of DTLS
   "records". These records are independent and can be processed
   correctly even in the face of loss or reordering. In DTLS-SRTP, this
   record protocol is replaced with SRTP [RFC3711]

Appendix B. Performance of Multiple DTLS Handshakes

   Standard practice for security protocols such as TLS, DTLS, and SSH,
   which do inline key management, is to create a separate security
   association for each underlying network channel (TCP connection, UDP
   host/port quartet, etc.). This has dual advantages of simplicity and
   independence of the security contexts for each channel.

   Three concerns have been raised about the overhead of this strategy
   in the context of RTP security. The first concern is the additional
   performance overhead of doing a separate public key operation for
   each channel. The conventional procedure here (used in TLS and DTLS)
   is to establish a master context that can then be used to derive
   fresh traffic keys for new associations. In TLS/DTLS, this is called
   "session resumption" and can be transparently negotiated between the
   peers.

   The second concern is network bandwidth overhead for the
   establishment of subsequent connections and for rehandshake (for
   rekeying) for existing connections. In particular, there is a
   concern that the channels will have very narrow capacity requirements
   allocated entirely to media that will be overflowed by the
   rehandshake. Measurements of the size of the rehandshake (with
   resumption) in TLS indicate that it is about 300-400 bytes if a full
   selection of cipher suites is offered. (The size of a full handshake
   is approximately 1-2 kilobytes larger because of the certificate and
   keying material exchange.)

   The third concern is the additional round-trips associated with
   establishing the second, third, ... channels. In TLS/DTLS, these can
   all be done in parallel, but in order to take advantage of session
   resumption they should be done after the first channel is
   established. For two channels, this provides a ladder diagram
   something like this (parenthetical numbers are media channel numbers)

```
    Alice                                     Bob
    -------------------------------------------
                        <-          ClientHello (1)
    ServerHello (1)     ->
    Certificate (1)
    ServerHelloDone (1)
                        <- ClientKeyExchange (1)
                           ChangeCipherSpec (1)
                                    Finished (1)
    ChangeCipherSpec (1)->
    Finished        (1)->

                                          <--- Channel 1 ready


                        <-          ClientHello (2)
    ServerHello (2)     ->
    ChangeCipherSpec(2)->
    Finished(2)         ->
                        <-  ChangeCipherSpec (2)
                                    Finished (2)
                                          <--- Channel 2 ready
```

                  Figure 6: Parallel DTLS-SRTP negotiations.

   So, there is an additional 1 RTT (round-trip time) after Channel 1 is
   ready before Channel 2 is ready. If the peers are potentially
   willing to forego resumption, they can interlace the handshakes, like
   so:

```
    Alice                                 Bob
    -------------------------------------------
                        <-        ClientHello (1)
    ServerHello (1)     ->
    Certificate (1)
    ServerHelloDone (1)
                        <- ClientKeyExchange (1)
                            ChangeCipherSpec (1)
                                     Finished (1)
                        <-        ClientHello (2)
    ChangeCipherSpec (1)->
    Finished        (1)->
                                             <--- Channel 1 ready
    ServerHello (2)     ->
    ChangeCipherSpec(2)->
    Finished(2)         ->
                        <-  ChangeCipherSpec (2)
                                     Finished (2)
                                             <--- Channel 2 ready
```

                Figure 7: Interlaced DTLS-SRTP negotiations.

   In this case, the channels are ready contemporaneously, but if a
   message in handshake (1) is lost, then handshake (2) requires either
   a full rehandshake or that Alice be clever and queue the resumption
   attempt until the first handshake completes. Note that just dropping
   the packet works as well, since Bob will retransmit.

Authors' Addresses

   David McGrew
   Cisco Systems
   510 McCarthy Blvd.
   Milpitas, CA 95305
   USA

   EMail: mcgrew@cisco.com

   Eric Rescorla
   RTFM, Inc.
   2064 Edgewood Drive
   Palo Alto, CA 94303
   USA

   EMail: ekr@rtfm.com