

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 20, 2008

D. McGrew  
Cisco Systems  
E. Rescorla  
Network Resonance  
November 17, 2007

Datagram Transport Layer Security (DTLS) Extension to Establish Keys for  
Secure Real-time Transport Protocol (SRTP)  
[draft-ietf-avt-dtls-srtp-01.txt](#)

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on May 20, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).

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.

Internet-Draft

SRTP Extension for DTLS

November 2007

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction . . . . .</a>	<a href="#">3</a>
<a href="#">2.</a>	<a href="#">Conventions Used In This Document . . . . .</a>	<a href="#">3</a>
<a href="#">3.</a>	<a href="#">Overview of DTLS-SRTP Operation . . . . .</a>	<a href="#">4</a>
<a href="#">4.</a>	<a href="#">DTLS Extensions for SRTP Key Establishment . . . . .</a>	<a href="#">5</a>
<a href="#">4.1.</a>	<a href="#">The use_srtp Extension . . . . .</a>	<a href="#">5</a>
<a href="#">4.1.1.</a>	<a href="#">use_srtp Extension Definition . . . . .</a>	<a href="#">6</a>
<a href="#">4.1.2.</a>	<a href="#">SRTP Protection Profiles . . . . .</a>	<a href="#">7</a>
<a href="#">4.1.3.</a>	<a href="#">srtp_mki value . . . . .</a>	<a href="#">9</a>
<a href="#">4.2.</a>	<a href="#">Key Derivation . . . . .</a>	<a href="#">10</a>
<a href="#">4.3.</a>	<a href="#">Key Scope . . . . .</a>	<a href="#">12</a>
<a href="#">4.4.</a>	<a href="#">Key Usage Limitations . . . . .</a>	<a href="#">12</a>
<a href="#">5.</a>	<a href="#">Use of RTP and RTCP over a DTLS-SRTP Channel . . . . .</a>	<a href="#">12</a>
<a href="#">5.1.</a>	<a href="#">Data Protection . . . . .</a>	<a href="#">12</a>
<a href="#">5.1.1.</a>	<a href="#">Transmission . . . . .</a>	<a href="#">12</a>
<a href="#">5.1.2.</a>	<a href="#">Reception . . . . .</a>	<a href="#">13</a>
<a href="#">5.2.</a>	<a href="#">Rehandshake and Re-key . . . . .</a>	<a href="#">13</a>
<a href="#">6.</a>	<a href="#">Multi-party RTP Sessions . . . . .</a>	<a href="#">14</a>
<a href="#">6.1.</a>	<a href="#">SIP Forking . . . . .</a>	<a href="#">14</a>
<a href="#">7.</a>	<a href="#">Security Considerations . . . . .</a>	<a href="#">15</a>
<a href="#">7.1.</a>	<a href="#">Security of Negotiation . . . . .</a>	<a href="#">15</a>
<a href="#">7.2.</a>	<a href="#">Framing Confusion . . . . .</a>	<a href="#">15</a>
<a href="#">7.3.</a>	<a href="#">Sequence Number Interactions . . . . .</a>	<a href="#">15</a>
<a href="#">7.3.1.</a>	<a href="#">Alerts . . . . .</a>	<a href="#">15</a>
<a href="#">7.3.2.</a>	<a href="#">Renegotiation . . . . .</a>	<a href="#">16</a>
<a href="#">8.</a>	<a href="#">IANA Considerations . . . . .</a>	<a href="#">16</a>
<a href="#">9.</a>	<a href="#">Acknowledgments . . . . .</a>	<a href="#">16</a>
<a href="#">10.</a>	<a href="#">References . . . . .</a>	<a href="#">17</a>
<a href="#">10.1.</a>	<a href="#">Normative References . . . . .</a>	<a href="#">17</a>
<a href="#">10.2.</a>	<a href="#">Informational References . . . . .</a>	<a href="#">17</a>
<a href="#">Appendix A.</a>	<a href="#">Performance of Multiple DTLS Handshakes . . . . .</a>	<a href="#">18</a>
	<a href="#">Authors' Addresses . . . . .</a>	<a href="#">20</a>
	<a href="#">Intellectual Property and Copyright Statements . . . . .</a>	<a href="#">21</a>

## [1.](#) Introduction

The Secure RTP profile (SRTP) [[6](#)] 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) [[5](#)] is a channel security protocol that offers integrated key management, parameter negotiation, and secure data transfer. Because DTLS's 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, an SRTP extension for DTLS which combine the performance and encryption flexibility benefits of SRTP with the flexibility and convenience of DTLS's 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 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 8](#) 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 [\[1\]](#).

McGrew & Rescorla

Expires May 20, 2008

[Page 3]

---

Internet-Draft

SRTP Extension for DTLS

November 2007

## [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 an 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. 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 [\[9\]](#) 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 [\[9\]](#) 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.

RTP and RTCP traffic MAY be multiplexed on a single UDP port [\[7\]](#). 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. It is RECOMMENDED that symmetric RTP is used, with RTP and RTCP multiplexed on a single UDP port; this requires only a single DTLS-SRTP session.

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, and how to authenticate the endpoints using fingerprints can be found in [\[12\]](#) and [\[11\]](#).

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 connection is established, and all other DTLS sessions wait until that connection is established before proceeding with their session establishment exchanges. This strategy allows the later sessions to use the DTLS session re-start, which allows the amortization of the expensive public key cryptography operations over multiple DTLS session establishment instances.

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 use the same SRTP keys, and similarly, all of the RTP sources originating on the server over the same channel use the same SRTP keys. The SRTP implementation MUST ensure that all of the SSRC values for all of the RTP sources originating from the same device are distinct, in order to avoid the "two-time pad" problem (as described in [Section 9.1 of](#)

[RFC 3711](#)).

## 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 and RTCP [3]. 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

Note that '\*' indicates messages which are not always sent in DTLS. The CertificateRequest, client Certificate, 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 [[10](#)] 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 need to distinguish them. The SRTP stack can use the procedures of [[10](#)] to distinguish RTP from RTCP.

When the "use\_srtp" extension is in effect, implementations MUST NOT place more than one "record" per datagram. (This is only meaningful from the perspective of DTLS because SRTP is inherently oriented towards one payload per packet, but is stated purely for clarification.)

Records of type other than "application\_data" MUST use ordinary DTLS framing.

#### [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 8](#) for the registration):

```
uint8 SRTPProtectionProfile[2];

struct {
    SRTPProtectionProfiles SRTPProtectionProfiles;
    uint8 srtp_mki<0..255>;
} UseSRTPData;

SRTPProtectionProfile SRTPProtectionProfiles<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 MasterKeyIdentifier (MKI) value (if any) which the client will use for his SRTP messages. If this field is of zero length, then no MKI will be used.

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 which the server has chosen for use with this connection. The server MUST NOT select a value which 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_SHA1_80 = {0x00, 0x01};
SRTPProtectionProfile SRTP_AES128_CM_SHA1_32 = {0x00, 0x02};
SRTPProtectionProfile SRTP_AES256_CM_SHA1_80 = {0x00, 0x03};
SRTPProtectionProfile SRTP_AES256_CM_SHA1_32 = {0x00, 0x04};
SRTPProtectionProfile SRTP_NULL_SHA1_80      = {0x00, 0x05};
SRTPProtectionProfile SRTP_NULL_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.



for the AES256\_CM cipher, which is specified in [14].

```
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_AES256_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_AES256_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 Key Derivation Function (KDF) is used to generate keys to feed into the SRTP KDF.
- o The Key Derivation Rate (KDR) is equal to zero. Thus, keys are not re-derived based on the SRTP sequence number.
- o For all other parameters, the default values are used.

All SRTP parameters that are not determined by the SRTP Protection Profile MAY be established via the signaling system. In particular, the relative order of Forward Error Correction and SRTP processing, and a suggested SRTP replay window size SHOULD be established in this manner. An example of how these parameters can be defined for SDP by is contained in [8]. If they are not otherwise signalled, they take on their default values from [7].

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 case 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 by [RFC 2434](#) Standards Action. See [Section 8](#) 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 Indicator (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.

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 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.

#### [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 extractor [\[15\]](#) to generate  $2 * (\text{SRTPSecurityParams.master\_key\_len} + \text{SRTPSecurityParams.master\_salt\_len})$  bytes of data, which are assigned as shown below.

```
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 extractor label for this usage is "dtls\_srtp".

The four keying material values are provided as inputs to the SRTP key derivation mechanism, as shown in Figure 5 and detailed below. By default, the mechanism defined in Section 4.3 of [\[7\]](#) 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.

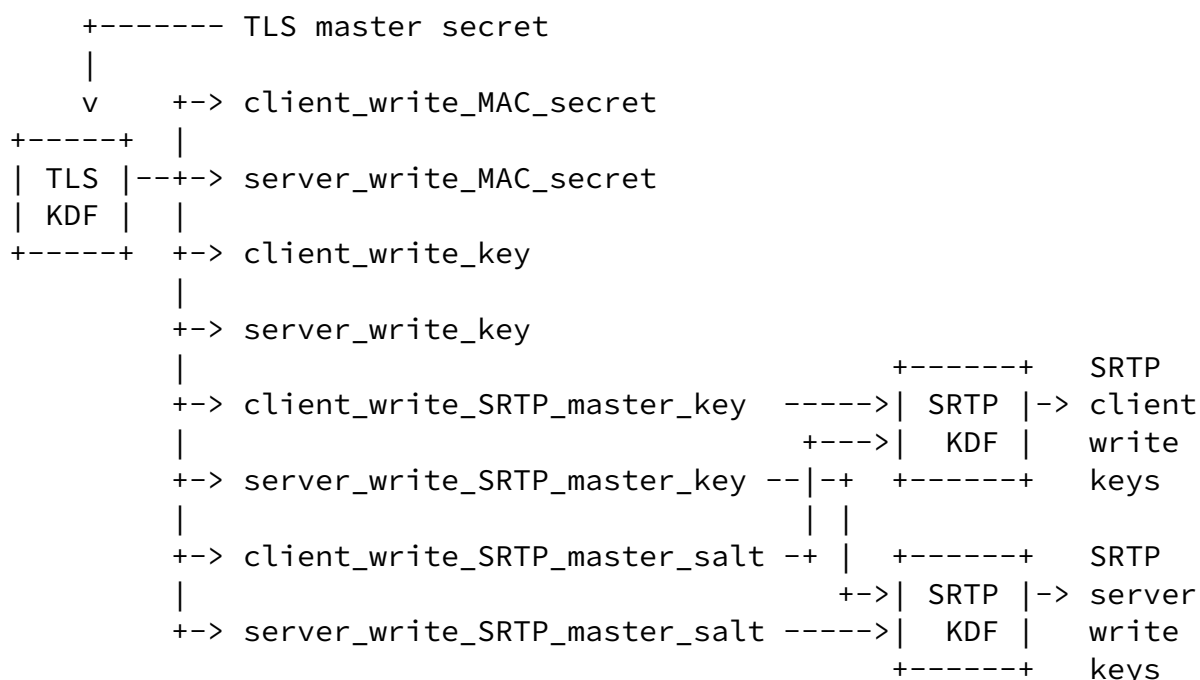


Figure 5: The derivation of the SRTP keys.

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 6. 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 (Sender)		Server (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 6: 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 re-key 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 of type "application\_data" (used to transport data traffic) 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 a single write of type "application\_data". 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[13]. 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 processes is summarized in Figure 7.

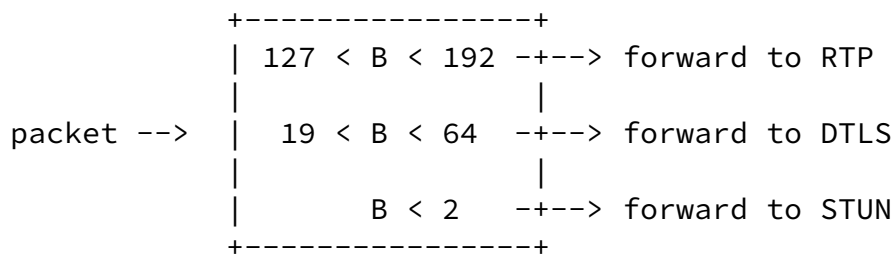


Figure 7: The DTLS-SRTP receiver's packet demultiplexing algorithm. Here the field B denotes the leading byte of the packet.

## 5.2. Rehandshake and Re-key

Rekeying in DTLS is accomplished by performing a new handshake over the existing DTLS channel. 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. 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 RTP flow in which there are exactly two participants. 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 which would allow 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.

### 6.1. SIP Forking

When SIP parallel forking occurs while establishing an RTP flow a situation may arise in which two or more sources are sending RTP packets to a single RTP destination transport address. When this situation arises and DTLS-SRTP is in use, the receiver **MUST** use the source transport IP address and port of each packet to distinguish between the senders, and treat the flow of packets from each distinct source transport address as a distinct DTLS-SRTP session for the purposes of the DTLS association.

When SIP forking occurs, the following method can be used to correlate each answer to the corresponding DTLS-SRTP session. If the answers have different certificates then fingerprints in the answers can be used to correlate the SIP dialogs with the associated DTLS session. Note that two forks with the same certificate cannot be

distinguished at the DTLS level, but this problem is a generic problem with SIP forking and should be solved at a higher level.



## [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 any attacker who can control the channel can always mount.

### [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 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 [\[4\]](#)).

### [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 which 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.

## [8](#). IANA Considerations

This document a new extension for DTLS, in accordance with [\[6\]](#):

```
enum { use_srtp (??) } ExtensionType;
```

`[[ NOTE: This value needs to be assigned by IANA ]]`

This extension MUST only be used with DTLS, and not with TLS.

[Section 4.1.2](#) requires that all SRTPProtectionProfile values be defined by [RFC 2434](#) Standards Action. IANA SHOULD create a DTLS SRTPProtectionProfile registry initially populated with values from [Section 4.1.2](#) of this document. Future values MUST be allocated via Standards Action as described in [\[2\]](#)

## [9](#). Acknowledgments

Special thanks to Flemming Andreassen, Francois Audet, Jason Fischl, Cullen Jennings, Colin Perkins, and Dan Wing, for input, discussions, and guidance.

Internet-Draft

SRTP Extension for DTLS

November 2007

## [10.](#) References

### [10.1.](#) Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [2] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 2434](#), October 1998.
- [3] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, [RFC 3550](#), July 2003.
- [4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", [RFC 4346](#), April 2006.
- [5] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", [RFC 4347](#), April 2006.
- [6] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", [RFC 4366](#), April 2006.
- [7] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", [RFC 3711](#), March 2004.
- [8] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", [RFC 4568](#), July 2006.
- [9] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", [BCP 131](#), [RFC 4961](#), July 2007.

### [10.2.](#) Informational References

- [10] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port",

[draft-ietf-avt-rtp-and-rtcp-mux-07](#) (work in progress),  
August 2007.

- [11] Fischl, J., "Datagram Transport Layer Security (DTLS) Protocol for Protection of Media Traffic Established with the Session Initiation Protocol", [draft-fischl-sipping-media-dtls-03](#) (work in progress), July 2007.

McGrew & Rescorla

Expires May 20, 2008

[Page 17]

---

Internet-Draft

SRTP Extension for DTLS

November 2007

- [12] Fischl, J. and H. Tschofenig, "Session Description Protocol (SDP) Indicators for Datagram Transport Layer Security (DTLS)", [draft-fischl-mmusic-sdp-dtls-03](#) (work in progress), July 2007.
- [13] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)", [draft-ietf-behave-rfc3489bis-13](#) (work in progress), November 2007.
- [14] McGrew, D., "The use of AES-192 and AES-256 in Secure RTP", [draft-mcgrew-srtp-big-aes-00](#) (work in progress), April 2006.
- [15] Rescorla, E., "Keying Material Extractors for Transport Layer Security (TLS)", [draft-rescorla-tls-extractor-00](#) (work in progress), January 2007.

## [Appendix A.](#) 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 which 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 which 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 1-2k larger because of the certificate and keying material exchange).

The third concern is the additional round-trips associated with establishing the 2nd, 3rd, ... 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 #s are media channel #s)

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

So, there is an additional 1 RTT 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
<hr/>		
	<-	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

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](mailto:mcgrew@cisco.com)

Eric Rescorla  
Network Resonance  
2064 Edgewood Drive  
Palo Alto, CA 94303  
USA

Email: [ekr@networkresonance.com](mailto:ekr@networkresonance.com)

#### Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS

OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).

## Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).