Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2007

D. McGrew
Cisco Systems
E. Rescorla
Network Resonance
March 10, 2007

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

draft-mcgrew-tls-srtp-02.txt

Status of this Memo

Copyright Notice

Abstract

The Secure Real-time Transport Protocol (SRTP) is a profile of the Real-time Transport Protocol that can provide confidentiality, message authentication, and replay protection to the RTP traffic and to the control traffic for RTP, the Real-time Transport Control Protocol (RTCP). This document describes a method of using DTLS key management for SRTP by using a new extension that indicates that SRTP

is to be used for data protection, and which establishes SRTP keys.

Table of Contents

1. Introduction

The Secure Real-time Transport Protocol (SRTP) [6] is a profile of the Real-time Transport Protocol (RTP) [1] that can provide confidentiality, message authentication, and replay protection to RTP traffic and to the control traffic for RTP, the Real-time Transport Control Protocol (RTCP). SRTP does not provide key management functionality but instead depends on external key management to provide secret master keys and 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.

This extension MUST only be used when the data being transported is RTP and RTCP [4].

The key points of DTLS-SRTP are that:

- application data is protected using SRTP,
- the DTLS handshake is used to establish keying material, algorithms, and parameters for SRTP,
- a DTLS extension used to negotiate SRTP algorithms, and
- other DTLS record layer content types are protected using the ordinary DTLS record format.

The next section provides details of the new extension.

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 [2].

3. Protocol Description

In this section we define the DTLS extension and its use.

3.1. Usage Model

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.

If both RTCP and RTP use the same source and destination ports [7], then the both the RTCP packets and the RTP packets are protected by a single DTLS-SRTP session. Otherwise, each RTCP flow is protected by a separate DTLS-SRTP session that is independent from the DTLS-SRTP session that protects the RTP packet flow.

Symmetric RTP 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.

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!). One or both of the DTLS-SRTP session participants MAY be RTP mixers.

A DTLS-SRTP session can 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 [9] and [8].

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

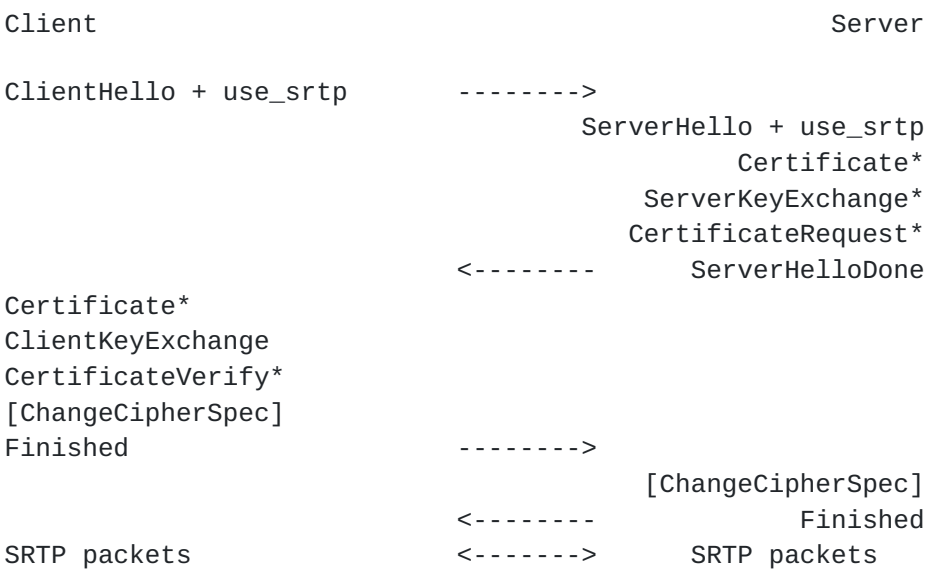
session establishment instances.

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).

3.2. The use_srtp Extension

In order to negotiate the use of SRTP data protection, clients MAY include an extension of type "use_srtp" in the extended client hello. 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 MAY 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.



Once the "use_srtp" extension is negotiated, packets of type "application_data" in the newly negotiated association (i.e., after the change_cipher_spec) MUST be protected using SRTP and packets of type "application_data" MUST NOT be sent. Records of type other than "application_data" MUST use ordinary DTLS framing. 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.)

3.2.1. use_srtp Extension Definition

The client MUST fill the extension_data field of the "use_srtp" extension with an UseSRTPData value:

```
uint8 SRTPProtectionProfile[2];

struct {
   SRTPProtectionProfiles SRTPProtectionProfiles;
   uint8 srtp_mki<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 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.

3.2.2. 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};


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.

All of the crypto algorithms in these profiles are from [6], except 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:

- The TLS Key Derivation Function (KDF) is used to generate keys to feed into the SRTP KDF.
- The Key Derivation Rate (KDR) is equal to zero. Thus, keys are not re-derived based on the SRTP sequence number.
- 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 [10].

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 Section 6 for IANA Considerations.

3.2.3. 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.

3.3. 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 as additional keying material at the end of the DTLS key block. Thus, the key block becomes:

```
client_write_MAC_secret[SecurityParameters.hash_size]
server_write_MAC_secret[SecurityParameters.hash_size]
client_write_key[SecurityParameters.key_material_len]
server_write_key[SecurityParameters.key_material_len]
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]
```

NOTE: It would probably be more attractive to use a TLS extractor as defined in [15]. However, this technique has not yet been vetted by

the TLS WG and therefore this remains an open issue.

The last four 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 [6] 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.

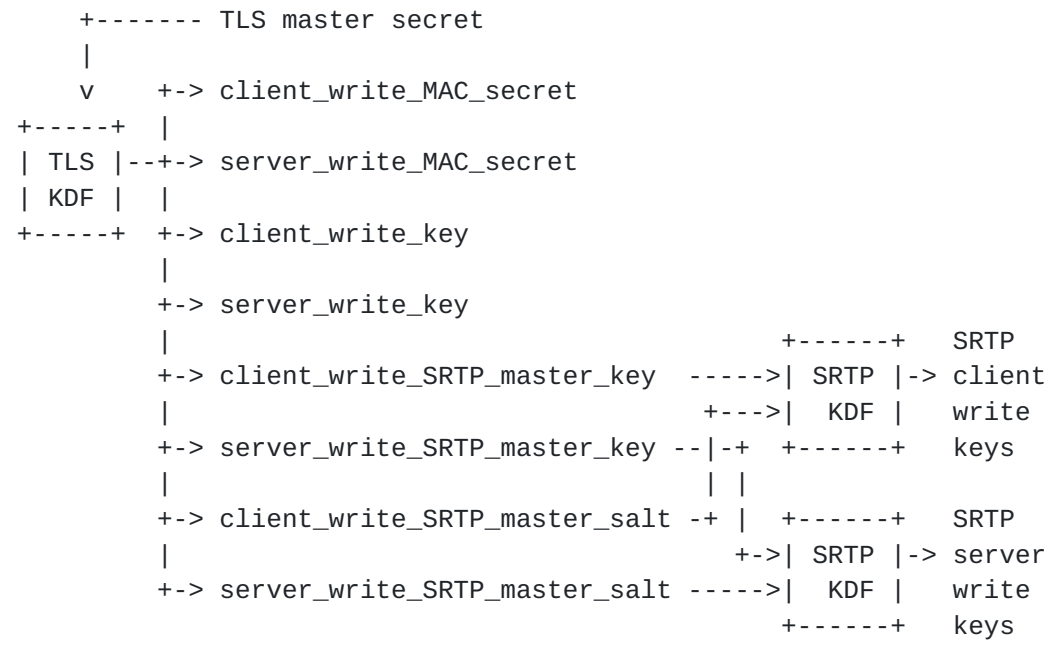


Figure 5: The derivation of the SRTP keys.

When both RTCP and RTP use the same source and destination ports, then both the SRTP and SRTCP keys are need. 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.

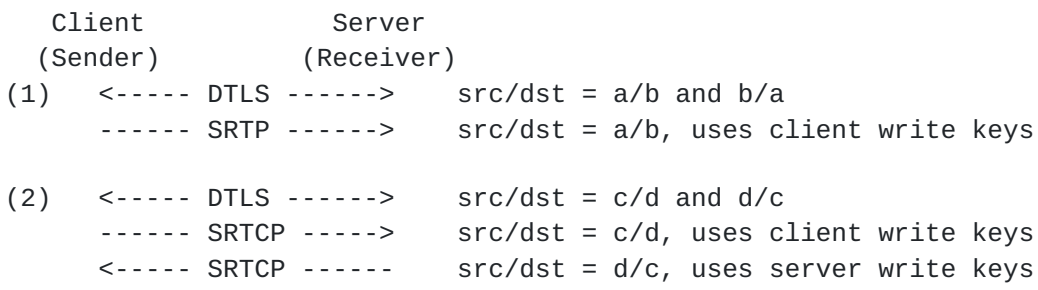


Figure 6: A DTLS-SRTP session protecting RTP (1) and another one protecting RTCP (2), showing the transport addresses and keys used.

3.4. 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.

3.5. 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.

3.6. 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.

3.6.1. 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.

3.6.2. 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.

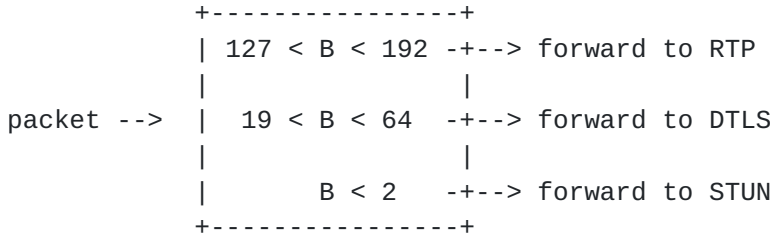


Figure 7: The DTLS-SRTP receiver's packet demultiplexing algorithm. Here the field B denotes the leading byte of the packet.

3.6.2.1. Opportunistic Probing

[[Open Issue In discussions of media-level security, some have suggested that a desirable property is to allow the endpoints to automatically detect the capability to do security at the media layer without interaction from the signalling. This issue is primarily out of scope for this document, however because of the demuxing mentioned above, it is possible for an implementation to "probe" by sending DTLS handshake packets and seeing if they are answered. A DTLS-SRTP implementation can demux the packets, detect that a handshake has been requested and notify the application to potentially initiate a DTLS-SRTP association. It is, however, necessary to have a rule to break the symmetry as to which side is client and which server. In applications where the media channel is established via SDP, the offeror should be the server and the answerer the client.]]

3.7. 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.

4. Multi-party RTP Sessions

Since DTLS is a point-to-point protocol, DTLS-SRTP is intended only to protect RTP sessions 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.

4.1. SIP Forking

When SIP parallel forking occurs while establishing an RTP session, 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.

5. Security Considerations

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

5.1. 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.

5.2. 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 [11]).

5.3. Sequence Number Interactions

As described in Section Section 3.6.1, 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.

5.3.1. 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.

5.3.2. Renegotiation

Because the rehandshake transition algorithm specified in Section Section 3.7 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.

6. IANA Considerations

This document a new extension for DTLS, in accordance with [12]:

```
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 Section 3.2.2 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 Section 3.2.2 of this document. Future values MUST be allocated via Standards Action as described in [3]

7. Acknowledgments

Special thanks to Jason Fischl, Flemming Andreasen, Dan Wing, and Cullen Jennings for input, discussions, and guidance.

8. References

8.1. Normative References

[1] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

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

[3] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

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

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

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

8.2. Informational References

[7] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", draft-ietf-avt-rtp-and-rtcp-mux-03 (work in progress), December 2006.

[8] Fischl, J., "Datagram Transport Layer Security (DTLS) Protocol for Protection of Media Traffic Established with the Session Initiation Protocol", draft-fischl-sipping-media-dtls-01 (work in progress), June 2006.

[9] Fischl, J. and H. Tschofenig, "Session Description Protocol (SDP) Indicators for Datagram Transport Layer Security (DTLS)", draft-fischl-mmusic-sdp-dtls-01 (work in progress), June 2006.

[10] Andreasen, F., "Session Description Protocol Security Descriptions for Media Streams", draft-ietf-mmusic-sdescriptions-12 (work in progress), September 2005.

[11] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.1", draft-ietf-tls-rfc2246-bis-13 (work in progress), June 2005.

[12] Blake-Wilson, S., "Transport Layer Security (TLS) Extensions", draft-ietf-tls-rfc3546bis-02 (work in progress), October 2005.

[13] Rosenberg, J., "Simple Traversal Underneath Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-05 (work in progress), October 2006.

[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.

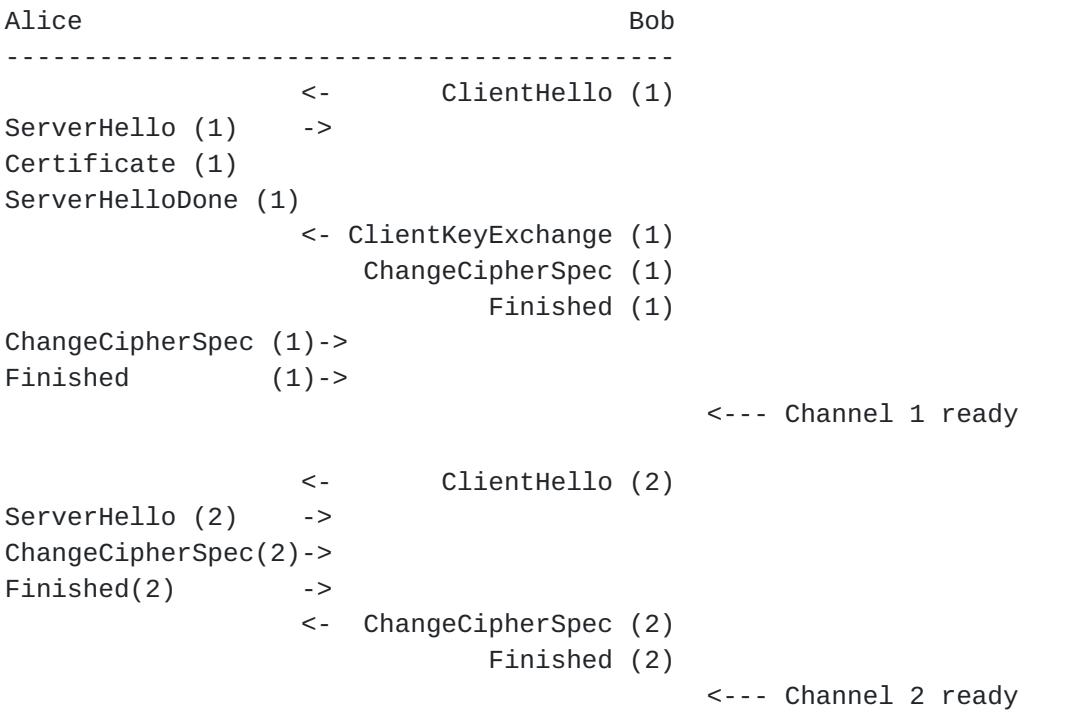
Appendix A. Open Issue: Key/Stream Interaction

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. Similar techniques could be applied to other inline RTP security protocols.

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)



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
-----------------------------------------
                  <-       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.

We don't know if this is a problem yet or whether it is possible to use some of the capacity allocated to other channels (e.g., RTCP) to perform the rehandshake. Another alternative that has been proposed is to use one security association connection on a single channel and reuse the keying material across multiple channels, but this gives up the simplicity and independence benefits mentioned above and so is architecturally undesirable unless absolutely necessary.

Another alternative is to take advantage of the fact that an (S)RTP channel is intended to be paired with an (S)RTCP channel. The DTLS handshake could be performed on just one of those channels and the same keys used for both the RTP and RTCP channels. This alternative is defined in Appendix B for study and discussion.

Appendix B. Open Issue: Using a single DTLS session per SRTP session

In order to address the performance, bandwidth, and latency concerns

described in Appendix A, it may be desirable to use a single DTLS session for each SRTP session. This appendix outlines one approach for achieving that goal in the next subsection, and then describes its benefits in the following one. However, note that the use of symmetric RTP and the multiplexing of RTCP and RTP over a single port pair largely eliminates this issue, since it allows a bidirectional pair of SRTP sessions (complete with SRTP and SRTCP) to be established following a single DTLS handshake.

B.1. Definition

This section defines the "Single DTLS" model by descibing its differences from the DTLS-SRTP as defined in the body of this document. A point-to-point SRTP session consists of a unidirectional SRTP flow and a bidirectional SRTCP flow. In the Single-DTLS model, a DTLS-SRTP session contains a single SRTP session and a single DTLS connection. Within each SRTP session, there may be multiple SRTP sources; all of these sources use a single SRTP master key. See Figure 11. As before, the DTLS connection uses the same transport addresses as the RTP flow; receivers demultiplex packets by inspection of their first byte.

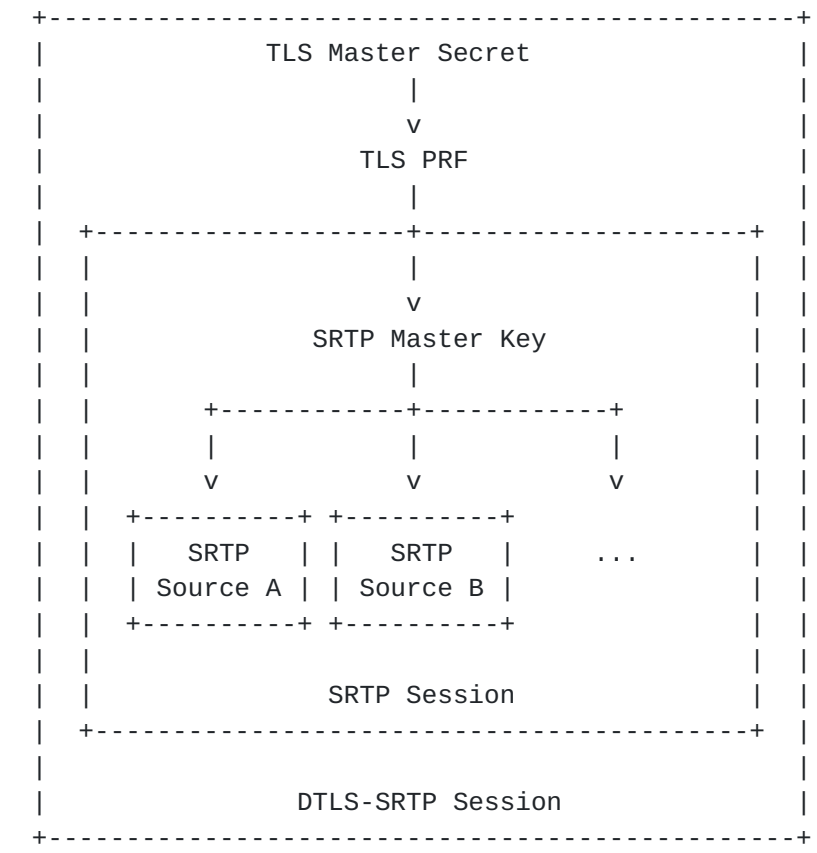


Figure 11: Key derivation in the Single-DTLS model.

Until the SRTP keys have been established by the DTLS handshake, the participants MUST reject all SRTP and SRTCP packets that they receive in the DTLS-SRTP session.

The SRTP keys are derived as defined in Section 3.3, and are used as shown in Figure 12. This figure should be contrasted with Figure 6.

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

Figure 12: A DTLS-SRTP session in the Single-DTLS model.

B.1.1. SRTP Parameter Profiles for Single-DTLS

The following list indicates the SRTP transform parameters for each protection profile in the Single-DTLS model. The main difference is that, in this model, an SRTP Parameter Profile determines the policy for the protection of RTP and RTCP packets.

```
SRTP_AES128_CM_HMAC_SHA1_80
      SRTP and SRTCP cipher:  AES_128_CM
      SRTP and SRTCP cipher_key_length:  128
      SRTP and SRTCP cipher_salt_length:  112
      SRTP maximum_lifetime:  2^48
      SRTCP maximum_lifetime:  2^31
      SRTP and SRTCP auth_function:  HMAC-SHA1
      SRTP and SRTCP auth_key_length:  160
      SRTP and SRTCP auth_tag_length:  80
SRTP_AES128_CM_HMAC_SHA1_32
      SRTP and SRTCP cipher:  AES_128_CM
      SRTP and SRTCP cipher_key_length:  128
      SRTP and SRTCP cipher_salt_length:  112
      SRTP maximum_lifetime:  2^48
      SRTCP maximum_lifetime:  2^31
      SRTP and SRTCP auth_function:  HMAC-SHA1
      SRTP and SRTCP auth_key_length:  160
      SRTP auth_tag_length:  32
      SRTCP auth_tag_length:  80
SRTP_AES256_CM_HMAC_SHA1_80
      SRTP and SRTCP cipher:  AES_128_CM
```

```
      SRTP and SRTCP cipher_key_length:  128
      SRTP and SRTCP cipher_salt_length:  112
      SRTP maximum_lifetime:  2^48
      SRTCP maximum_lifetime:  2^31
      SRTP and SRTCP auth_function:  HMAC-SHA1
      SRTP and SRTCP auth_key_length:  160
      SRTP and SRTCP auth_tag_length:  80
SRTP_AES256_CM_HMAC_SHA1_32
      SRTP and SRTCP cipher:  AES_128_CM
      SRTP and SRTCP cipher_key_length:  128
      SRTP and SRTCP cipher_salt_length:  112
      SRTP maximum_lifetime:  2^48
      SRTCP maximum_lifetime:  2^31
      SRTP and SRTCP auth_function:  HMAC-SHA1
      SRTP and SRTCP auth_key_length:  160
      SRTP auth_tag_length:  32
      SRTCP auth_tag_length:  80
SRTP_NULL_HMAC_SHA1_80
      SRTP and SRTCP cipher:  NULL
      SRTP and SRTCP cipher_key_length:  0
      SRTP and SRTCP cipher_salt_length:  0
      SRTP maximum_lifetime:  2^48
      SRTCP maximum_lifetime:  2^31
      SRTP and SRTCP auth_function:  HMAC-SHA1
      SRTP and SRTCP auth_key_length:  160
      SRTP and SRTCP auth_tag_length:  80
SRTP_NULL_HMAC_SHA1_32
      SRTP and SRTCP cipher:  NULL
      SRTP and SRTCP cipher_key_length:  0
      SRTP and SRTCP cipher_salt_length:  0
      SRTP maximum_lifetime:  2^48
      SRTCP maximum_lifetime:  2^31
      SRTP and SRTCP auth_function:  HMAC-SHA1
      SRTP and SRTCP auth_key_length:  160
      SRTP auth_tag_length:  32
      SRTCP auth_tag_length:  80
```

B.2. Pros and Cons of Single-DTLS

Using a single DTLS session per SRTP session has potential performance benefits in terms of reducing latency and compution. The discussion of the performance of multiple parallel DTLS connections in Appendix A applies here as well. In addition, it provides a good match for existing SRTP implementations, since it matches their SRTP policy definitions for cryptographic algorithm configuration and makes use of all of the derived keys rather than having to discard half as described in Section 3.3.

Authors' Addresses

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

Email: mcgrew@cisco.com

Eric Rescorla
Network Resonance
2483 E. Bayshore #212
Palo Alto, CA 94303
USA

Email: ekr@networkresonance.com

Full Copyright Statement

Intellectual Property

Acknowledgment

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