

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

C. Jennings
P. Jones
Cisco Systems
A. Roach
Mozilla
March 21, 2016

S RTP Double Encryption Procedures
draft-jennings-perc-double-01

Abstract

In some conferencing scenarios, it is desirable for an intermediary to be able to manipulate some RTP parameters, while still providing strong end-to-end security guarantees. This document defines SRTP procedures that use two separate but related cryptographic contexts to provide "hop-by-hop" and "end-to-end" security guarantees. Both the end-to-end and hop-by-hop cryptographic transforms can utilize an authenticated encryption with associated data scheme or take advantage of future SRTP transforms with different properties.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Cryptographic Contexts	3
4. Original Header Block	4
5. RTP Operations	5
5.1. Encrypting a Packet	5
5.2. Modifying a Packet	6
5.3. Decrypting a Packet	7
6. RTCP Operations	8
7. Recommended Inner and Outer Cryptographic Transforms	8
8. Security Considerations	9
9. IANA Considerations	10
9.1. RTP Header Extension	10
9.2. DTLS-SRTP	10
10. Acknowledgments	12
11. References	12
11.1. Normative References	12
11.2. Informative References	12
Authors' Addresses	13

1. Introduction

Cloud conferencing systems that are based on switched conferencing have a central media distribution device (MDD) that receives media from endpoints and distributes it to other endpoints, but does not need to interpret or change the media content. For these systems, it is desirable to have one cryptographic context from the sending endpoint to the receiving endpoint that can encrypt and authenticate the media end-to-end while still allowing certain RTP header information to be changed by the MDD. At the same time, a separate cryptographic context provides integrity and optional confidentiality for the media flowing between the MDD and the endpoints. See the framework document that describes this concept in more detail in more detail in [I-D.jones-perc-private-media-framework].

This specification RECOMMENDS the SRTP AES-GCM transform [RFC7714] to encrypt an RTP packet for the end-to-end cryptographic context. The output of this is treated as an RTP packet and again encrypted with an SRTP transform used in the hop-by-hop cryptographic context between the endpoint and the MDD. The MDD decrypts and checks

integrity of the hop-by-hop security. The MDD MAY change some of the RTP header information that would impact the end-to-end integrity. The original value of any RTP header field that is changed is included in a new RTP header extension called the Original Header Block. The new RTP packet is encrypted with the hop-by-hop cryptographic transform before it is sent. The receiving endpoint decrypts and checks integrity using the hop-by-hop cryptographic transform and then replaces any parameters the MDD changed using the information in the Original Header Block before decrypting and checking the end-to-end integrity.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Terms used throughout this document include:

- o MDD: media distribution device that routes media from one endpoint to other endpoints
- o E2E: end-to-end, meaning the link from one endpoint through one or more MDDs to the endpoint at the other end.
- o HBH: hop-by-hop, meaning the link from the endpoint to or from the MDD.
- o OHB: Original Header Block is an RTP header extension that contains the original values from the RTP header that might have been changed by an MDD.

3. Cryptographic Contexts

This specification uses two cryptographic contexts: an inner ("end-to-end") context that is used by endpoints that originate and consume media to ensure the integrity of media end-to-end, and an outer ("hop-by-hop") context that is used between endpoints and MDDs to ensure the integrity of media over a single hop and to enable an MDD to modify certain RTP header fields. RTCP is also encrypted using the hop-by-hop cryptographic context. The RECOMMENDED cipher for the hop-by-hop and end-to-end contexts is AES-GCM. Other combinations of SRTP ciphers that support the procedures in this document can be added to the IANA registry.

The keys and salt for these contexts are generated with the following steps:

- o Generate key and salt values of the length required for the combined inner (end-to-end) and outer (hop-by-hop) transforms.
- o Assign the key and salt values generated for the inner (end-to-end) transform.
- o Assign the key and salt values for the outer (hop-by-hop) transform.

As a special case, the outer cryptographic transform MAY be the NULL cipher (see [RFC3711]) if a secure transport such as [RFC6347] is used over a hop (i.e., between an endpoint and MDD or between two MDDs). In that case, the key and salt values generated would be the length required only for the inner cryptographic transform.

Obviously, if the MDD is to be able to modify header fields but not decrypt the payload, then it must have cryptographic context for the outer transform, but not the inner transform. This document does not define how the MDD should be provisioned with this information. One possible way to provide keying material for the outer ("hop-by-hop") transform is to use [I-D.jones-perc-dtls-tunnel].

4. Original Header Block

Any SRTP packet processed following these procedures MAY contain an Original Header Block (OHB) RTP header extension.

The OHB contains the original values of any modified header fields and MUST be placed after any already-existing RTP header extensions. Placement of the OHB after any original header extensions is important so that the receiving endpoint can properly authenticate the original packet and any originally included RTP header extensions. The receiving endpoint will authenticate the original packet by restoring the modified RTP header field values and header extensions. It does this by copying the original values from the OHB and then removing the OHB extension and any other RTP header extensions that appear after the OHB extension.

The MDD is only permitted to modify the extension (X) bit, payload type (PT) field, and the RTP sequence number field.

The OHB extension is either one octet in length, two octets in length, or three octets in length. The length of the OHB indicates what data is contained in the extension.

If the OHB is one octet in length, it contains both the original X bit and PT field value. In this case, the OHB has this form:

```

0
0 1 2 3 4 5 6 7
+-----+
|X|      PT      |
+-----+

```

If the OHB is two octets in length, it contains the original RTP packet sequence number. In this case, the OHB has this form:

```

0                               1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-----+
|      Sequence Number      |
+-----+

```

If the OHB is three octets in length, it contains the original X bit, PT field value, and RTP packet sequence number. In this case, the OHB has this form:

```

0                               1                               2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-----+-----+-----+
|X|      PT      |      Sequence Number      |
+-----+-----+-----+

```

If an MDD modifies an original RTP header value, the MDD MUST include the OHB extension to reflect the changed value. If another MDD along the media path makes additional changes to the RTP header and any original value is not already present in the OHB, the MDD must extend the OHB by adding the changed value to the OHB. To properly preserve original RTP header values, an MDD MUST NOT change a value already present in the OHB extension.

5. RTP Operations

5.1. Encrypting a Packet

To encrypt a packet, the endpoint encrypts the packet using the inner cryptographic context and then encrypts using the outer cryptographic context. The processes is as follows:

- o Form an RTP packet. If there are any header extensions, they MUST use [RFC5285].
- o Apply the inner cryptographic transform to the RTP packet. If encrypting RTP header extensions end-to-end, then [RFC6904] MUST be used when encrypting the RTP packet using the inner cryptographic context.

- o If the endpoint wishes to insert header extensions that can be modified by an MDD, it MUST insert an OHB header extension at the end of any header extensions protected end-to-end, then add any MDD-modifiable header extensions. The OHB MUST replicate the information found in the RTP header following the application of the inner cryptographic transform. For example, if the packet had no header extensions when the inner cryptographic transform was applied, the X bit would be 0. If the endpoint introduces an OHB and then adds MDD-modifiable header extensions, the X bit in the OHB would be 0. After introducing the OHB and MDD-modifiable header extensions, of course, the X bit in the RTP header would be set to 1.
- o Apply the outer cryptographic transform to the RTP packet. If encrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used when encrypting the RTP packet using the outer cryptographic context.

5.2. Modifying a Packet

The MDD does not have a notion of outer or inner cryptographic contexts. Rather, the MDD has a single cryptographic context. The cryptographic transform and key used to decrypt a packet and any encrypted RTP header extensions would be the same as those used in the endpoint's outer cryptographic context.

In order to modify a packet, the MDD decrypts the packet, modifies the packet, updates the OHB with any modifications not already present in the OHB, and re-encrypts the packet using the cryptographic context used for next hop.

- o Apply the cryptographic transform to the packet. If decrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used.
- o Change any required parameters
- o If a changed RTP header field is not already in the OHB, add it with its original value to the OHB. An MDD MAY add information to the OHB, but MUST NOT change existing information in the OHB.
- o If the MDD resets a parameter to its original value, it MAY drop it from the OHB as long as there are no other header extensions following the OHB. Note that this might result in a decrease in the size of the OHB.
- o The MDD MUST NOT delete any header extensions before the OHB, but MAY add, delete, or modify any that follow the OHB.

- * If the MDD adds any header extensions, it must append them and it must maintain the order of the original header extensions in the [RFC5285] block.
- * If the MDD appends header extensions, then it MUST add the OHB header extension (if not present), even if the OHB merely replicates the original header field values, and append the new extensions following the OHB. The OHB serves as a demarcation point between original RTP header extensions introduced by the endpoint and those introduced by an MDD.
- o The MDD MAY modify any header extension appearing after the OHB, but MUST NOT modify header extensions that are present before the OHB.
- o Apply the cryptographic transform to the packet. If encrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used.

5.3. Decrypting a Packet

To decrypt a packet, the endpoint first decrypts and verifies using the outer cryptographic context, then uses the OHB to reconstruct the original packet, which it decrypts and verifies with the inner cryptographic context.

- o Apply the outer cryptographic transform to the packet. If the integrity check does not pass, discard the packet. The result of this is referred to as the outer SRTP packet. If decrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used when decrypting the RTP packet using the outer cryptographic context.
- o Form a new synthetic SRTP packet with:
 - * Header = Received header, with header fields replaced with values from OHB (if present).
 - * Insert all header extensions up to the OHB extension, but exclude the OHB and any header extensions that follow the OHB. If the original X bit is 1, then the remaining extensions MUST be padded to the first 32-bit boundary and the overall length of the header extensions adjusted accordingly. If the original X bit is 0, then the header extensions would be removed entirely.
 - * Payload is the original encrypted payload.
- o Apply the inner cryptographic transform to this synthetic SRTP packet. If the integrity check does not pass, discard the packet.

If decrypting RTP header extensions end-to-end, then [RFC6904] MUST be used when decrypting the RTP packet using the inner cryptographic context.

Once the packet has successfully decrypted, the application needs to be careful about which information it uses to get the correct behavior. The application MUST use only the information found in the synthetic SRTP packet and MUST NOT use the other data that was in the outer SRTP packet with the following exceptions:

- o The PT from the outer SRTP packet is used for normal matching to SDP and codec selection.
- o The sequence number from the outer SRTP packet is used for normal RTP ordering.

If any of the following RTP headers extensions are found in the outer SRTP packet, they MAY be used:

- o TBD

6. RTCP Operations

Unlike RTP, which is encrypted both hop-by-hop and end-to-end using two separate cryptographic contexts, RTCP is encrypted using only the outer (HBH) cryptographic context. The procedures for RTCP encryption are specified in [RFC3711] and this document introduces no additional steps.

7. Recommended Inner and Outer Cryptographic Transforms

This specification recommends and defines AES-GCM as both the inner and outer cryptographic transforms, identified as DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM and DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM. These transforms provide for authenticated encryption and will consume additional processing time double-encrypting for HBH and E2E. However, the approach is secure and simple, and is thus viewed as an acceptable trade-off in processing efficiency.

Note that names for the cryptographic transforms are of the form DOUBLE_(inner transform)_(outer transform).

This specification also allows for the NULL cipher to be used as the outer cryptographic transform in cases where a secure transport is used over the hop, with those transforms identified as DOUBLE_AEAD_AES_128_GCM_NULL_NULL and DOUBLE_AEAD_AES_256_GCM_NULL_NULL.

Open Issue: It is not clear if the NULL ciphers are needed or not. The authors plan to remove them from the next version of the draft unless there is a reasonable support and reasons to keep them in.

While this document only defines a profile based on AES-GCM, it is possible for future documents to define further profiles with different inner and outer transforms in this same framework. For example, if a new SRTP transform was defined that encrypts some or all of the RTP header, it would be reasonable for systems to have the option of using that for the outer transform. Similarly, if a new transform was defined that provided only integrity, that would also be reasonable to use for the HBH as the payload data is already encrypted by the E2E.

The AES-GCM cryptographic transform introduces an additional 16 octets to the length of the packet. When using AES-GCM for both the inner and outer cryptographic transforms, the total additional length is 32 octets. If no other header extensions are present in the packet and the OHB is introduced, that will consume an additional 8 octets. If other extensions are already present, the OHB will consume up to 4 additional octets.

8. Security Considerations

It is obviously critical that the intermediary have only the outer transform parameters and not the inner transform parameters. We rely on an external key management protocol to assure this property.

Modifications by the intermediary result in the recipient getting two values for changed parameters (original and modified). The recipient will have to choose which to use; there is risk in using either that depends on the session setup.

The security properties for both the inner and outer key holders are the same as the security properties of classic SRTP.

The NULL cipher MUST be used in conjunction with an encrypted transport for both RTP and RTCP. Use of the NULL cipher for the outer cryptographic context without the use of an encrypted transport exposes the RTCP traffic to undetectable modification as it is transmitted over the network. Likewise, RTP traffic under the same conditions would be subject to modification that would not be detectable by the MDD. While the endpoint could detect modification of the end-to-end information, reliance on information like payload type value in the packet received from the MDD could present problems such as attempting to decode media with the wrong codec.

9. IANA Considerations

9.1. RTP Header Extension

This document defines a new extension URI in the RTP Compact Header Extensions part of the Real-Time Transport Protocol (RTP) Parameters registry, according to the following data:

Extension URI: urn:ietf:params:rtp-hdext:ohb

Description: Original Header Block

Contact: Cullen Jennings <mailto:fluffy@iii.ca>

Reference: RFCXXXX

Note to RFC Editor: Replace RFCXXXX with the RFC number of this specification.

9.2. DTLS-SRTP

We request IANA to add the following values to defines a DTLS-SRTP "SRTP Protection Profile" defined in [RFC5764].

Value	Profile	Reference
{TBD, TBD}	DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM	RFCXXXX
{TBD, TBD}	DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM	RFCXXXX
{TBD, TBD}	DOUBLE_AEAD_AES_128_GCM_NULL_NULL	RFCXXXX
{TBD, TBD}	DOUBLE_AEAD_AES_256_GCM_NULL_NULL	RFCXXXX

Note to IANA: Please assign value RFCXXXX and update table to point at this RFC for these values.

The SRTP transform parameters for each of these protection are:

DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM

cipher: AES_128_GCM then AES_128_GCM
cipher_key_length: 256 bits
cipher_salt_length: 192 bits
aead_auth_tag_length: 32 octets
auth_function: NULL
auth_key_length: N/A
auth_tag_length: N/A
maximum lifetime: at most 2^{31} SRTCP packets and
at most 2^{48} SRTP packets

DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM

cipher: AES_256_GCM then AES_256_GCM
cipher_key_length: 512 bits
cipher_salt_length: 192 bits
aead_auth_tag_length: 32 octets
auth_function: NULL
auth_key_length: N/A
auth_tag_length: N/A
maximum lifetime: at most 2^{31} SRTCP packets and
at most 2^{48} SRTP packets

DOUBLE_AEAD_AES_128_GCM_NULL_NULL

cipher: AES_128_GCM then identity transform
cipher_key_length: 128 bits
cipher_salt_length: 96 bits
aead_auth_tag_length: 16 octets
auth_function: NULL
auth_key_length: N/A
auth_tag_length: N/A
maximum lifetime: at most 2^{31} SRTCP packets and
at most 2^{48} SRTP packets

DOUBLE_AEAD_AES_256_GCM_NULL_NULL

cipher: AES_256_GCM then identity transform
cipher_key_length: 256 bits
cipher_salt_length: 96 bits
aead_auth_tag_length: 16 octets
auth_function: NULL
auth_key_length: N/A
auth_tag_length: N/A
maximum lifetime: at most 2^{31} SRTCP packets and
at most 2^{48} SRTP packets

Except when the NULL cipher is used for the outer (HBH) transform, the first half of the key and salt is used for the inner (E2E) transform and the second half is used for the outer (HBH) transform.

For those that use the NULL cipher for the outer transform, the the key and salt is applied only to the inner transform.

10. Acknowledgments

Many thanks to review from GET YOUR NAME HERE. Please, send comments.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<http://www.rfc-editor.org/info/rfc3711>>.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, DOI 10.17487/RFC5285, July 2008, <<http://www.rfc-editor.org/info/rfc5285>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<http://www.rfc-editor.org/info/rfc5764>>.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <<http://www.rfc-editor.org/info/rfc6904>>.
- [RFC7714] McGrew, D. and K. Igoe, "AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)", RFC 7714, DOI 10.17487/RFC7714, December 2015, <<http://www.rfc-editor.org/info/rfc7714>>.

11.2. Informative References

[I-D.jones-perc-dtls-tunnel]

Jones, P., "DTLS Tunnel between Media Distribution Device and Key Management Function to Facilitate Key Exchange", draft-jones-perc-dtls-tunnel-02 (work in progress), March 2016.

[I-D.jones-perc-private-media-framework]

Jones, P., Ismail, N., and D. Benham, "A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing", draft-jones-perc-private-media-framework-02 (work in progress), March 2016.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

Authors' Addresses

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca

Paul E. Jones
Cisco Systems

Email: paulej@packetizer.com

Adam Roach
Mozilla

Email: adam@nostrum.com

PERC Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 22, 2016

J. Mattsson, Ed.
Ericsson
D. McGrew
D. Wing
F. Andreassen
C. Jennings
Cisco
April 20, 2016

Encrypted Key Transport for Secure RTP
draft-jennings-perc-srtp-ekt-diet-01

Abstract

IMPORTANT: This draft is just a cut down version of draft-ietf-avtcore-srtp-ekt-03 to help discussion about the key parts of EKT for the PERC WG. Any changes decided here would need to be synchronized with the draft-ietf-avtcore-srtp-ekt draft. Nearly all the text here came from draft-ietf-avtcore-srtp-ekt and the authors of that draft.

Encrypted Key Transport (EKT) is an extension to Secure Real-time Transport Protocol (SRTP) that provides for the secure transport of SRTP master keys, Rollover Counters, and other information within SRTP. This facility enables SRTP to work for decentralized conferences with minimal control by allowing a common key to be used across multiple endpoints.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Conventions Used In This Document	4
2. Encrypted Key Transport	4
2.1. EKT Field Formats	4
2.2. Packet Processing and State Machine	6
2.2.1. Outbound Processing	6
2.2.2. Inbound Processing	7
2.3. Ciphers	8
2.3.1. The Default Cipher	9
2.3.2. Other EKT Ciphers	9
2.4. Synchronizing Operation	10
2.5. Transport	10
2.6. Timing and Reliability Consideration	10
3. Use of EKT with DTLS-SRTP	11
3.1. DTLS-SRTP Recap	11
3.2. EKT Extensions to DTLS-SRTP	11
3.3. Offer/Answer Considerations	13
4. Sending the DTLS EKT_Key Reliably	13
5. Security Considerations	13
6. Open Issues	14
7. IANA Considerations	15
8. Acknowledgements	15
9. References	15
9.1. Normative References	15
9.2. Informative References	16
Authors' Addresses	16

1. Introduction

RTP is designed to allow decentralized groups with minimal control to establish sessions, such as for multimedia conferences. Unfortunately, Secure RTP (SRTP [RFC3711]) cannot be used in many minimal-control scenarios, because it requires that SSRC values and other data be coordinated among all of the participants in a session. For example, if a participant joins a session that is already in progress, that participant needs to be told the SRTP keys (and SSRC, ROC and other details) of the other SRTP sources.

The inability of SRTP to work in the absence of central control was well understood during the design of the protocol; the omission was considered less important than optimizations such as bandwidth conservation. Additionally, in many situations SRTP is used in conjunction with a signaling system that can provide most of the central control needed by SRTP. However, there are several cases in which conventional signaling systems cannot easily provide all of the coordination required. It is also desirable to eliminate the layer violations that occur when signaling systems coordinate certain SRTP parameters, such as SSRC values and ROCs.

This document defines Encrypted Key Transport (EKT) for SRTP and reduces the amount of external signaling control that is needed in a SRTP session that is shared with multiple receivers. EKT securely distributes the SRTP master key and other information for each SRTP source. With this method, SRTP entities are free to choose SSRC values as they see fit, and to start up new SRTP sources (SSRC) with new SRTP master keys (see Section 2.2) within a session without coordinating with other entities via external signaling or other external means.

EKT provides a way for an SRTP session participant, either a sender or receiver, to securely transport its SRTP master key and current SRTP rollover counter to the other participants in the session. This data furnishes the information needed by the receiver to instantiate an SRTP/SRTCP receiver context.

EKT does not control the manner in which the SSRC is generated; it is only concerned with their secure transport.

EKT is not intended to replace external key establishment mechanisms. Instead, it is used in conjunction with those methods, and it relieves them of the burden of tightly coordinating every SRTP source (SSRC) among every SRTP participant.

1.1. 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].

2. Encrypted Key Transport

EKT defines a new method of providing SRTP master keys to an endpoint. In order to convey the ciphertext of the SRTP master key, and other additional information, an additional EKT field is added to SRTP packets. When added to SRTP, the EKT field appears at the end of the SRTP packet, after the authentication tag (if that tag is present), or after the ciphertext of the encrypted portion of the packet otherwise.

EKT MUST NOT be used in conjunction with SRTP's MKI (Master Key Identifier) or with SRTP's <From, To> [RFC3711], as those SRTP features duplicate some of the functions of EKT.

2.1. EKT Field Formats

The EKT Field uses the format defined below for the Full_EKT_Field and Short_EKT_Field.

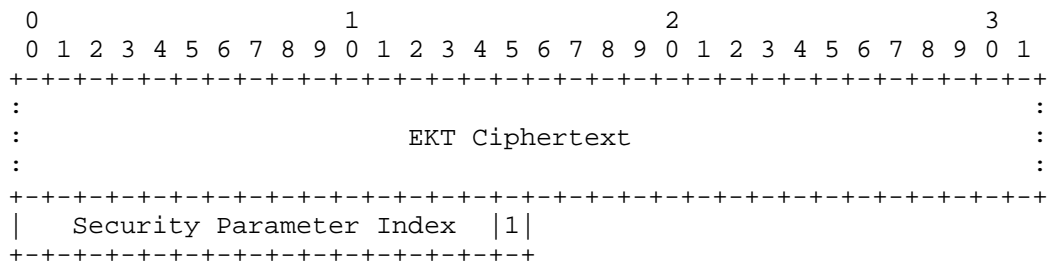


Figure 1: Full EKT Field format

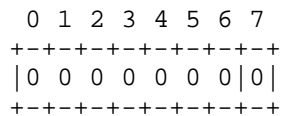


Figure 2: Short EKT Field format

```
EKT_Plaintext = SRTP_Master_Key || SSRC || ROC  
EKT_Ciphertext = EKT_Encrypt(EKT_Key, EKT_Plaintext)  
Full_EKT_Field = EKT_Ciphertext || SPI || '1'  
Short_EKT_Field = '00000000'
```

Figure 3: EKT data formats

These fields and data elements are defined as follows:

EKT_Plaintext: The data that is input to the EKT encryption operation. This data never appears on the wire, and is used only in computations internal to EKT. This is the concatenation of the SRTP Master Key, the SSRC, and the ROC.

EKT_Ciphertext: The data that is output from the EKT encryption operation, described in Section 2.3. This field is included in SRTP packets when EKT is in use. The length of this field is variable, and is equal to the ciphertext size N defined in Section 2.3. Note that the length of the field is inferable from the SPI field, since the SPI will indicate the cipher being used and thus the size.

SRTP_Master_Key: On the sender side, the SRTP Master Key associated with the indicated SSRC. The length of this field depends on the cipher suite negotiated during call setup for SRTP or SRTCP.

SSRC: On the sender side, this field is the SSRC for this SRTP source. The length of this field is 32 bits.

Rollover Counter (ROC): On the sender side, this field is set to the current value of the SRTP rollover counter in the SRTP context associated with the SSRC in the SRTP or SRTCP packet. The length of this field is 32 bits.

Security Parameter Index (SPI): This field indicates the appropriate EKT Key and other parameters for the receiver to use when processing the packet. Each time a different EKT Key is received, it will have a different SPI. The length of this field is 15 bits. The parameters identified by this field are:

- * The EKT cipher used to process the packet.
- * The EKT Key used to process the packet.

- * The SRTP Master Salt associated with any Master Key encrypted with this EKT Key.

Together, these data elements are called an EKT parameter set. Within each SRTP session, each distinct EKT parameter set that may be used MUST be associated with a distinct SPI value, to avoid ambiguity.

Final bit: The last bit is used to indicate the type of the Field. This MUST be 1 in the Full EKT Field format and 0 in Short EKT Field.

2.2. Packet Processing and State Machine

At any given time, each SRTP/SRTCP source (SSRC) has associated with it a single EKT parameter set. This parameter set is used to process all outbound packets, and is called the outbound parameter set for that SSRC. There may be other EKT parameter sets that are used by other SRTP/SRTCP sources in the same session, including other SRTP/SRTCP sources on the same endpoint (e.g., one endpoint with voice and video might have two EKT parameter sets, or there might be multiple video sources on an endpoint each with their own EKT parameter set). All of the received EKT parameter sets SHOULD be stored by all of the participants in an SRTP session, for use in processing inbound SRTP and SRTCP traffic.

All SRTP master keys MUST NOT be re-used, MUST be randomly generated according to [RFC4086], and MUST NOT be equal to or derived from other SRTP master keys.

Either the Full_EKT_Field or Short_EKT_Field is appended at the tail end of all the SRTP packet.

2.2.1. Outbound Processing

See Section 2.6 which describes when to send an EKT packet with a Full EKT Field. If a Full EKT Field is not being sent, then a Short EKT Field needs to be sent so the receiver can correctly determine how to process the packet.

When an SRTP packet is to be sent with a Full EKT Field, the EKT field for that packet is created as follows, or uses an equivalent set of steps. The creation of the EKT field MUST precede the normal SRTP packet processing.

1. The Security Parameter Index field is set to the value of the Security Parameter Index that is associated with the outbound parameter set.

2. The EKT_Plaintext field is computed from the SRTP Master Key, SSRC, and ROC fields, as shown in Section 2.1. The ROC, SRTP Master Key, and SSRC used in EKT processing SHOULD be the same as the one used in the SRTP processing.
3. The EKT_Ciphertext field is set to the ciphertext created by encrypting the EKT_Plaintext with the EKT cipher, using the EKT Key as the encryption key. The encryption process is detailed in Section 2.3.
4. Then the Full EKT Field is formed using the EKT Ciphertext and the SPI associated with the EKT Key used above. The computed value of the Full EKT Field is written into the packet.

When a packet is sent with the Short EKT Field, the Short EKF Field is simply appended to the packet.

2.2.2. Inbound Processing

Inbound EKT processing MUST take place prior to the usual SRTP or SRTCP processing. The following steps show processing as packets are received in order.

1. The final bit is checked to determine which EKT format is in use. When an SRTP or SRTCP packet contains a Short EKT Field, the Short EKT Field is removed from the packet then normal SRTP or SRTCP processing occurs. If the packet contains a Full EKT Field, then processing continues as described below.
2. The combination of the SSRC and the Security Parameter Index (SPI) field is used to find which EKT parameter set should be used when processing the packet. If there is no matching SPI, then the verification function MUST return an indication of authentication failure, and the steps described below are not performed. EKT parameter set contains the EKT Key, EKT Cipher, and SRTP Master Salt.
3. The EKT Ciphertext authentication is checked and it is decrypted, as described in Section 2.3, using the EKT Key and EKT Cipher found in the previous step. If the EKT decryption operation returns an authentication failure, then the packet processing stops.
4. The resulting EKT Plaintext is parsed as described in Section 2.1, to recover the SRTP Master Key, SSRC, and ROC fields. The Master Salt that is associated with the EKT Keys used to do the decryption is also retrieved.

5. The SRTP Master Key, ROC, and Master Salt from the previous step are saved in a map indexed by the SSRC found in the EKT Plaintext and used for any future inbound or about crypto operations on packets with the that SSRC.
6. At this point, EKT processing has successfully completed, and the normal SRTP or SRTCP processing takes place including replay protection.

Implementation note: the receiver may want to have a sliding window to retain old SRTP master keys (and related context) for some brief period of time, so that out of order packets can be processed as well as packets sent during the time keys are changing.

2.3. Ciphers

EKT uses an authenticated cipher to encrypt and authenticate the EKT Plaintext. We first specify the interface to the cipher, in order to abstract the interface away from the details of that function. We then define the cipher that is used in EKT by default. The default cipher described in Section 2.3.1 MUST be implemented, but another cipher that conforms to this interface MAY be used, in which case its use MUST be coordinated by external means (e.g., key management).

An EKT cipher consists of an encryption function and a decryption function. The encryption function $E(K, P)$ takes the following inputs:

- o a secret key K with a length of L bytes, and
- o a plaintext value P with a length of M bytes.

The encryption function returns a ciphertext value C whose length is N bytes, where N is at least M . The decryption function $D(K, C)$ takes the following inputs:

- o a secret key K with a length of L bytes, and
- o a ciphertext value C with a length of N bytes.

The decryption function returns a plaintext value P that is M bytes long, or returns an indication that the decryption operation failed because the ciphertext was invalid (i.e. it was not generated by the encryption of plaintext with the key K).

These functions have the property that $D(K, E(K, P)) = P$ for all values of K and P . Each cipher also has a limit T on the number of times that it can be used with any fixed key value. For each key,

the encryption function MUST NOT be invoked on more than T distinct values of P, and the decryption function MUST NOT be invoked on more than T distinct values of C.

Security requirements for EKT ciphers are discussed in Section 5.

2.3.1. The Default Cipher

The default EKT Cipher is the Advanced Encryption Standard (AES) Key Wrap with Padding [RFC5649] algorithm. It requires a plaintext length M that is at least one octet, and it returns a ciphertext with a length of $N = M + 8$ octets. It can be used with key sizes of L = 16, and 32 octets, and its use with those key sizes is indicated as AESKW_128, or AESKW_256, respectively. The key size determines the length of the AES key used by the Key Wrap algorithm. With this cipher, $T=2^{48}$.

length of SRTP transform	length of EKT transform	EKT plaintext	EKT ciphertext	length of Full EKT Field
-----	-----	-----	-----	-----
AES-128	AESKW_128	26	40	42
AES-256	AESKW_256	42	56	58

Figure 4: AESKW Table

As AES-128 is the mandatory to implement transform in SRTP [RFC3711], AESKW_128 MUST be implemented for EKT.

For all the SRTP transforms listed in the table, the corresponding EKT transform MUST be used, unless a stronger EKT transform is negotiated by key management.

2.3.2. Other EKT Ciphers

Other specifications may extend this one by defining other EKT ciphers per Section 7. This section defines how those ciphers interact with this specification.

An EKT cipher determines how the EKT Ciphertext field is written, and how it is processed when it is read. This field is opaque to the other aspects of EKT processing. EKT ciphers are free to use this field in any way, but they SHOULD NOT use other EKT or SRTP fields as an input. The values of the parameters L, M, N, and T MUST be defined by each EKT cipher, and those values MUST be inferable from the EKT parameter set.

2.4. Synchronizing Operation

If a source has its EKT key changed by the key management, it MUST also change its SRTP master key, which will cause it to send out a new Full EKT Field. This ensures that if key management thought the EKT key needs changing (due to a participant leaving or joining) and communicated that in key management to a source, the source will also change its SRTP master key, so that traffic can be decrypted only by those who know the current EKT key.

2.5. Transport

EKT SHOULD be used over SRTP, and other specification MAY define how to use it over SRTCP. SRTP is preferred because it shares fate with transmitted media, because SRTP rekeying can occur without concern for RTCP transmission limits, and to avoid SRTCP compound packets with RTP translators and mixers.

2.6. Timing and Reliability Consideration

A system using EKT learns the SRTP master keys distributed with Full EKT Fields send with the SRTP, rather than with call signaling. A receiver can immediately decrypt an SRTP provided the SRTP packet contains a Full EKT Field.

This section describes how to reliably and expediently deliver new SRTP master keys to receivers.

There are three cases to consider. The first case is a new sender joining a session which needs to communicate its SRTP master key to all the receivers. The second case is a sender changing its SRTP master key which needs to be communicated to all the receivers. The third case is a new receiver joining a session already in progress which needs to know the sender's SRTP master key.

New sender: A new sender SHOULD send a packet containing the Full EKT Field as soon as possible, always before or coincident with sending its initial SRTP packet. To accommodate packet loss, it is RECOMMENDED that three consecutive packets contain the Full EKT Field be transmitted.

Rekey: By sending EKT over SRTP, the rekeying event shares fate with the SRTP packets protected with that new SRTP master key.

New receiver: When a new receiver joins a session it does not need to communicate its sending SRTP master key (because it is a receiver). When a new receiver joins a session the sender is generally unaware of the receiver joining the session. Thus, senders SHOULD

periodically transmit the Full EKT Field. That interval depends on how frequently new receivers join the session, the acceptable delay before those receivers can start processing SRTP packets, and the acceptable overhead of sending the Full EKT Field. The RECOMMENDED frequency is the same as the key frame frequency if sending video and every 100ms for audio.

3. Use of EKT with DTLS-SRTP

This document defines an extension to DTLS-SRTP called Key Transport. The EKT with the DTLS-SRTP Key Transport enables secure transport of EKT keying material from one DTLS-SRTP peer to another. This enables those peers to process EKT keying material in SRTP (or SRTCP) and retrieve the embedded SRTP keying material. This combination of protocols is valuable because it combines the advantages of DTLS (strong authentication of the endpoint and flexibility) with the advantages of EKT (allowing secure multiparty RTP with loose coordination and efficient communication of per-source keys).

3.1. DTLS-SRTP Recap

DTLS-SRTP [RFC5764] uses an extended DTLS exchange between two peers to exchange keying material, algorithms, and parameters for SRTP. The SRTP flow operates over the same transport as the DTLS-SRTP exchange (i.e., the same 5-tuple). DTLS-SRTP 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.

3.2. EKT Extensions to DTLS-SRTP

This document adds a new TLS negotiated extension called "ekt". This adds a new TLS content type, EKT, and a new negotiated extension EKT. The DTLS server includes "ekt" in its TLS ServerHello message. If a DTLS client includes "ekt" in its ClientHello, but does not receive "ekt" in the ServerHello, the DTLS client MUST NOT send DTLS packets with the "ekt" content-type.

Using the syntax described in DTLS [RFC6347], the following structures are used:

```
enum {
    ekt_key(0),
    ekt_key_ack(1),
    ekt_key_error(254),
    (255)
} SRTPKeyTransportType;

struct {
    SRTPKeyTransportType keytrans_type;
    uint24 length;
    uint16 message_seq;
    uint24 fragment_offset;
    uint24 fragment_length;
    select (SRTPKeyTransportType) {
        case ekt_key:
            EKTkey;
    };
} KeyTransport;

enum {
    RESERVED(0),
    AESKW_128(1),
    AESKW_256(3),
} ektcipher;

struct {
    ektcipher EKT_Cipher;
    uint EKT_Key_Value<1..256>;
    uint EKT_Master_Salt<1..256>;
    uint16 EKT_SPI;
} EKTkey;
```

Figure 5: Additional TLS Data Structures

The diagram below shows a message flow of DTLS client and DTLS server using the DTLS-SRTP Key Transport extension.

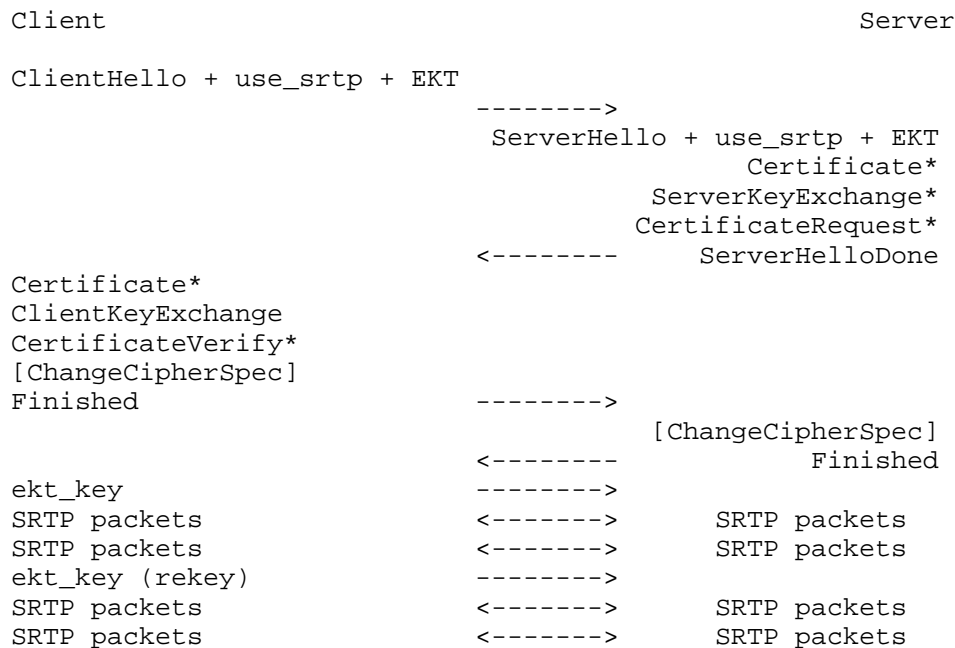


Figure 6: Handshake Message Flow

3.3. Offer/Answer Considerations

When using EKT with DTLS-SRTP, the negotiation to use EKT is done at the DTLS handshake level and does not change the [RFC3264] Offer / Answer messaging.

4. Sending the DTLS EKT_Key Reliably

The DTLS ekt_key is sent using the retransmissions specified in Section 4.2.4. of DTLS [RFC6347].

5. Security Considerations

EKT inherits the security properties of the DTLS-SRTP (or other) keying it uses.

With EKT, each SRTP sender and receiver MUST generate distinct SRTP master keys. This property avoids any security concern over the re-use of keys, by empowering the SRTP layer to create keys on demand. Note that the inputs of EKT are the same as for SRTP with key-sharing: a single key is provided to protect an entire SRTP session. However, EKT remains secure even when SSRC values collide.

The EKT Cipher includes its own authentication/integrity check. For an attacker to successfully forge a full EKT packet, it would need to defeat the authentication mechanisms of the EKT Cipher authentication mechanism.

The presence of the SSRC in the EKT_Plaintext ensures that an attacker cannot substitute an EKT_Ciphertext from one SRTP stream into another SRTP stream.

An attacker who tampers with the bits in Full_EKT_Field can prevent the intended receiver of that packet from being able to decrypt it. This is a minor denial of service vulnerability.

An attacker could send packets containing a Full EKT Field, in an attempt to consume additional CPU resources of the receiving system by causing the receiving system will decrypt the EKT ciphertext and detect an authentication failure

EKT can rekey an SRTP stream until the SRTP rollover counter (ROC) needs to roll over. EKT does not extend SRTP's rollover counter (ROC), and like SRTP itself EKT cannot properly handle a ROC rollover. Thus even if using EKT, new (master or session) keys need to be established after 2^{48} packets are transmitted in a single SRTP stream as described in Section 3.3.1 of [RFC3711]. Due to the relatively low packet rates of typical RTP sessions, this is not expected to be a burden.

The confidentiality, integrity, and authentication of the EKT cipher MUST be at least as strong as the SRTP cipher.

Part of the EKT_Plaintext is known, or easily guessable to an attacker. Thus, the EKT Cipher MUST resist known plaintext attacks. In practice, this requirement does not impose any restrictions on our choices, since the ciphers in use provide high security even when much plaintext is known.

An EKT cipher MUST resist attacks in which both ciphertexts and plaintexts can be adaptively chosen and adversaries that can query both the encryption and decryption functions adaptively.

6. Open Issues

What length should the SPI be?

Should we limit the number of saved SPI for a given SSRC? Or limit the lifetime of old ones after a new one is received? At some level this may not matter because even if the a SRTP packet is injected with an old value, it will be discarded by the RTP stack for being

old. It is more important that new things are encrypted with the most recent EKT Key.

How many bits to differentiate different types of packets and allow for extensibility?

Given the amount of old EKT deployed, should the Full EKT use a different code point than the "1" at the end?

Do we need AES-192?

7. IANA Considerations

No IANA actions are required.

8. Acknowledgements

Thanks to David Benham, Eddy Lem, Felix Wyss, Jonathan Lennox, Kai Fischer, Lakshminath Dondeti, Magnus Westerlund, Michael Peck, Nermeen Ismail, Paul Jones, Rob Raymond, and Yi Cheng for fruitful discussions, comments, and contributions to this document.

This draft is a cut down version of draft-ietf-avtcore-srtp-ekt-03 and most of the text here came from that draft.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<http://www.rfc-editor.org/info/rfc3711>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<http://www.rfc-editor.org/info/rfc4086>>.
- [RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <<http://www.rfc-editor.org/info/rfc5649>>.

- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<http://www.rfc-editor.org/info/rfc5764>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

9.2. Informative References

- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<http://www.rfc-editor.org/info/rfc3264>>.

Authors' Addresses

John Mattsson (editor)
Ericsson AB
SE-164 80 Stockholm
Sweden

Phone: +46 10 71 43 501
Email: john.mattsson@ericsson.com

David A. McGrew
Cisco Systems
510 McCarthy Blvd.
Milpitas, CA 95035
US

Phone: (408) 525 8651
Email: mcgrew@cisco.com
URI: <http://www.mindspring.com/~dmcgrew/dam.htm>

Dan Wing
Cisco Systems
510 McCarthy Blvd.
Milpitas, CA 95035
US

Phone: (408) 853 4197
Email: dwing@cisco.com

Flemming Andreason
Cisco Systems
499 Thornall Street
Edison, NJ 08837
US

Email: fandreas@cisco.com

Cullen Jennings
Cisco Systems
Calgary, AB
Canada

Email: fluffy@iii.ca

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

P. Jones
Cisco Systems
P. Ellenbogen
Princeton University
N. Ohlmeier
Mozilla
October 31, 2016

DTLS Tunnel between a Media Distributor and Key Distributor to
Facilitate Key Exchange
draft-jones-perc-dtls-tunnel-04

Abstract

This document defines a DTLS tunneling protocol for use in multimedia conferences that enables a Media Distributor to facilitate key exchange between an endpoint in a conference and the Key Distributor. The protocol is designed to ensure that the keying material used for hop-by-hop encryption and authentication is accessible to the media distributor, while the keying material used for end-to-end encryption and authentication is inaccessible to the media distributor.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Conventions Used In This Document	3
3. Tunneling Concept	3
4. Example Message Flows	4
5. Tunneling Procedures	6
5.1. Endpoint Procedures	6
5.2. Tunnel Establishment Procedures	6
5.3. Versioning Considerations	7
5.4. Media Distributor Tunneling Procedures	7
5.5. Key Distributor Tunneling Procedures	9
6. Tunneling Protocol	10
6.1. Tunnel Message Format	10
7. Example Binary Encoding	12
8. IANA Considerations	13
9. Security Considerations	13
10. Acknowledgments	14
11. References	14
11.1. Normative References	14
11.2. Informative References	15
Authors' Addresses	15

1. Introduction

An objective of the work in the Privacy-Enhanced RTP Conferencing (PERC) working group is to ensure that endpoints in a multimedia conference have access to the end-to-end (E2E) and hop-by-hop (HBH) keying material used to encrypt and authenticate Real-time Transport Protocol (RTP) [RFC3550] packets, while the Media Distributor has access only to the hop-by-hop (HBH) keying material for encryption and authentication.

This specification defines a tunneling protocol that enables the media distributor to tunnel DTLS [RFC6347] messages between an endpoint and the key distributor, thus allowing an endpoint to use DTLS-SRTP [RFC5764] for establishing encryption and authentication keys with the key distributor.

The tunnel established between the media distributor and key distributor is a TLS connection that is established before any

messages are forwarded by the media distributor on behalf of the endpoint. DTLS packets received from the endpoint are encapsulated by the media distributor inside this tunnel as data to be sent to the key distributor. Likewise, when the media distributor receives data from the key distributor over the tunnel, it extracts the DTLS message inside and forwards the DTLS message to the endpoint. In this way, the DTLS association for the DTLS-SRTP procedures is established between the endpoint and the key distributor, with the media distributor simply forwarding packets between the two entities and having no visibility into the confidential information exchanged.

Following the existing DTLS-SRTP procedures, the endpoint and key distributor will arrive at a selected cipher and keying material, which are used for HBH encryption and authentication by both the endpoint and the media distributor. However, since the media distributor would not have direct access to this information, the key distributor explicitly shares the HBH key information with the media distributor via the tunneling protocol defined in this document. Additionally, the endpoint and key distributor will agree on a cipher for E2E encryption and authentication. The key distributor will transmit keying material to the endpoint for E2E operations, but will not share that information with the media distributor.

By establishing this TLS tunnel between the media distributor and key distributor and implementing the protocol defined in this document, it is possible for the media distributor to facilitate the establishment of a secure DTLS association between an endpoint and the key distributor in order for the endpoint to receive E2E and HBH keying material. At the same time, the key distributor can securely provide the HBH keying material to the media distributor.

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] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

3. Tunneling Concept

A TLS connection (tunnel) is established between the media distributor and the key distributor. This tunnel is used to relay DTLS messages between the endpoint and key distributor, as depicted in Figure 1:

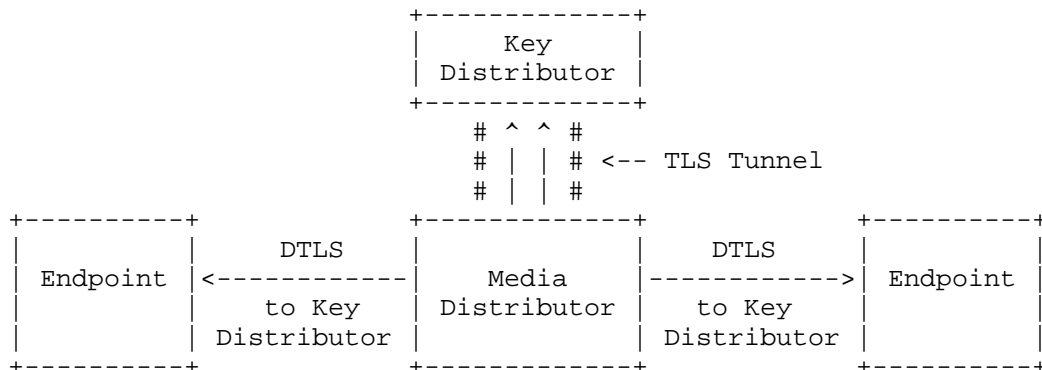


Figure 1: TLS Tunnel to Key Distributor

The three entities involved in this communication flow are the endpoint, the media distributor, and the key distributor. The behavior of each entity is described in Section 5.

The key distributor is a logical function that might be co-resident with a key management server operated by an enterprise, reside in one of the endpoints participating in the conference, or elsewhere that is trusted with E2E keying material.

4. Example Message Flows

This section provides an example message flow to help clarify the procedures described later in this document. It is necessary that the key distributor and media distributor establish a mutually authenticated TLS connection for the purpose of sending tunneled messages, though the complete TLS handshake for the tunnel is not shown in Figure 2 since there is nothing new this document introduces with regard to those procedures.

Once the tunnel is established, it is possible for the media distributor to relay the DTLS messages between the endpoint and the key distributor. Figure 2 shows a message flow wherein the endpoint uses DTLS-SRTP to establish an association with the key distributor. In the process, the media distributor shares its supported SRTP protection profile information (see [RFC5764]) and the key distributor shares HBH keying material and selected cipher with the media distributor.

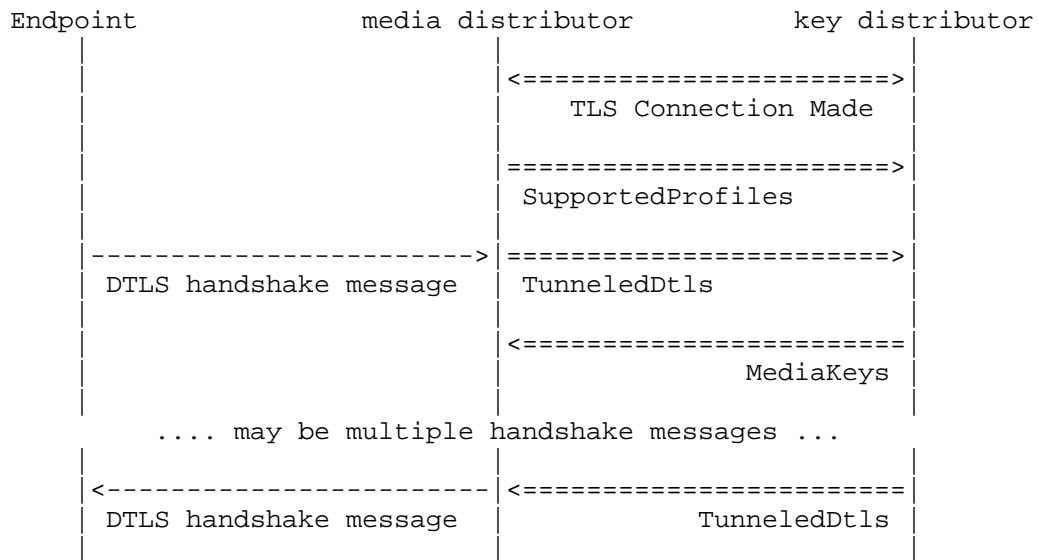


Figure 2: Sample DTLS-SRTP Exchange via the Tunnel

After the initial TLS connection has been established each of the messages on the right-hand side of Figure 2 is a tunneling protocol message as defined in Section 6.

SRTP protection profiles supported by the media distributor will be sent in a "SupportedProfiles" message when the TLS tunnel is initially established. The key distributor will use that information to select a common profile supported by both the endpoint and the media distributor to ensure that hop-by-hop operations can be successfully performed.

As DTLS messages are received from the endpoint by the media distributor, they are forwarded to the key distributor encapsulated inside abbrev "TunneledDtls" message. Likewise, as "TunneledDtls" messages are received by the media distributor from the key distributor, the encapsulated DTLS packet is forwarded to the endpoint.

The key distributor will provide the SRTP [RFC3711] keying material to the media distributor for HBH operations via the "MediaKeys" message. The media distributor will extract this keying material from the "MediaKeys" message when received and use it for hop-by-hop encryption and authentication.

5. Tunneling Procedures

The following sub-sections explain in detail the expected behavior of the endpoint, the media distributor, and the key distributor.

It is important to note that the tunneling protocol described in this document is not an extension to TLS [RFC5246] or DTLS [RFC6347]. Rather, it is a protocol that transports DTLS messages generated by an endpoint or key distributor as data inside of the TLS connection established between the media distributor and key distributor.

5.1. Endpoint Procedures

The endpoint follows the procedures outlined for DTLS-SRTP [RFC5764] in order to establish the cipher and keys used for encryption and authentication, with the endpoint acting as the client and the key distributor acting as the server. The endpoint does not need to be aware of the fact that DTLS messages it transmits toward the media distributor are being tunneled to the key distributor.

5.2. Tunnel Establishment Procedures

Either the media distributor or key distributor initiates the establishment of a TLS tunnel. Which entity acts as the TLS client when establishing the tunnel and what event triggers the establishment of the tunnel are outside the scope of this document. Further, how the trust relationships are established between the key distributor and media distributor are also outside the scope of this document.

A tunnel **MUST** be a mutually authenticated TLS connection.

The media distributor or key distributor **MUST** establish a tunnel prior to forwarding tunneled DTLS messages. Given the time-sensitive nature of DTLS-SRTP procedures, a tunnel **SHOULD** be established prior to the media distributor receiving a DTLS message from an endpoint.

A single tunnel **MAY** be used to relay DTLS messages between any number of endpoints and the key distributor.

A media distributor **MAY** have more than one tunnel established between itself and one or more key distributors. When multiple tunnels are established, which tunnel or tunnels to use to send messages for a given conference is outside the scope of this document.

5.3. Versioning Considerations

All messages for an established tunnel MUST utilize the same version value. If the version of any subsequent message differs from that of the initial message, that message MUST be discarded and the tunnel connection closed.

Since the media distributor sends the first message over the tunnel, it effectively establishes the version of the protocol to be used. If that version is not supported by the key distributor, it MUST discard the message, transmit an "UnsupportedVersion" message, and close the TLS connection.

The media distributor MUST take note of the version received in an "UnsupportedVersion" message and use that version when attempting to re-establish a failed tunnel connection. Note that it is not necessary for the media distributor to understand the newer version of the protocol to understand that the first message received is "UnsupportedVersion". The media distributor can determine from the first two octets received what the version number is and that the message is "UnsupportedVersion". The rest of the data received, if any, would be discarded and the connection closed (if not already closed).

5.4. Media Distributor Tunneling Procedures

The first message transmitted over the tunnel is the "SupportedProfiles" (see Section 6). This message informs the key distributor about which DTLS-SRTP profiles the media distributor supports. This message MUST be sent each time a new tunnel connection is established or, in the case of connection loss, when a connection is re-established.

The media distributor MUST forward all messages received from an endpoint for a given DTLS association through the same tunnel if more than one tunnel has been established between it and a key distributor.

Editor's Note: Do we want to have the above requirement or would we prefer to allow the media distributor to send messages over more than one tunnel to more than one key distributor? The latter would provide for higher availability, but at the cost of key distributor complexity. The former would allow the usage of a load distributor in front of the key distributor.

The media distributor MUST assign a unique association identifier for each endpoint-initiated DTLS association and include it in all messages forwarded to the key distributor. The key distributor will

subsequently include this identifier in all messages it sends so that the media distributor can map messages received via a tunnel and forward those messages to the correct endpoint. The association identifier SHOULD be randomly assigned and values not be re-used for a short period of time (e.g., five minutes) to ensure any residual state in the key distributor is clear and to ensure any packets already transmitted from the key distributor are not directed to the wrong endpoint.

The tunnel protocol enables the key distributor to separately provide HBH keying material to the media distributor for each of the individual endpoint DTLS associations, though the media distributor cannot decrypt messages between the key distributor and endpoints.

When a DTLS message is received by the media distributor from an endpoint, it forwards the UDP payload portion of that message to the key distributor encapsulated in a "TunneledDtls" message. If the media distributor knows which conference to which a given DTLS association belongs, it can pass the conference identifier to the key distributor using the "conf_id" field of the "TunneledDtls" message.

The media distributor MUST support the same list of protection profiles for the life of a given endpoint's DTLS association, which is represented by the association identifier.

When a "MediaKeys" message is received, the media distributor MUST extract the cipher and keying material conveyed in order to subsequently perform HBH encryption and authentication operations for RTP and RTCP packets sent between it and an endpoint. Since the HBH keying material will be different for each endpoint, the media distributor uses the association identifier included by the key distributor to ensure that the HBH keying material is used with the correct endpoint.

The media distributor MUST forward all DTLS messages received from either the endpoint or the key distributor (via the "TunneledDtls" message) to ensure proper communication between those two entities.

When the media distributor detects an endpoint has disconnected or when it receives conference control messages indicating the endpoint is to be disconnected, the media distributors MUST send an "EndpointDisconnect" message with the association identifier assigned to the endpoint to the key distributor. The media distributor SHOULD take a loss of all RTP and RTCP packets as an indicator that the endpoint has disconnected. The particulars of how RTP and RTCP are to be used to detect an endpoint disconnect, such as timeout period, is not specified. The media distributor MAY use additional indicators to determine when an endpoint has disconnected.

5.5. Key Distributor Tunneling Procedures

When the media distributor relays a DTLS message from an endpoint, the media distributor will include an association identifier that is unique per endpoint-originated DTLS association. The association identifier remains constant for the life of the DTLS association. The key distributor identifies each distinct endpoint-originated DTLS association by the association identifier.

The key distributor MUST encapsulate any DTLS message it sends to an endpoint inside a "TunneledDtls" message (see Section 6).

The key distributor MUST use the same association identifier in messages sent to an endpoint as was received in messages from that endpoint. This ensures the media distributor can forward the messages to the correct endpoint.

The key distributor extracts tunneled DTLS messages from an endpoint and acts on those messages as if that endpoint had established the DTLS association directly with the key distributor. The key distributor is acting as the DTLS server and the endpoint is acting as the DTLS client. The handling of the messages and certificates is exactly the same as normal DTLS-SRTP procedures between endpoints.

The key distributor MUST send a "MediaKeys" message to the media distributor as soon as the HBH encryption key is computed and before it sends a DTLS "Finished" message to the endpoint. The "MediaKeys" message includes the selected cipher (i.e. protection profile), MKI [RFC3711] value (if any), SRTP master keys, and SRTP master salt values. The key distributor MUST use the same association identifier in the "MediaKeys" message as is used in the "TunneledDtls" messages for the given endpoint.

The key distributor, can use the certificate of the endpoint and correlate that with signaling information to know which conference this session is associated with. The key distributor informs the media distributor of which conference this session is associated by sending a globally unique conference identifier in the "conf_id" attribute of the "MediaKeys".

The key distributor MUST select a cipher that is supported by both the endpoint and the media distributor to ensure proper HBH operations.

6. Tunneling Protocol

Tunneled messages are transported via the TLS tunnel as application data between the media distributor and the key distributor. Tunnel messages are specified using the format described in [RFC5246] section 4. As in [RFC5246], all values are stored in network byte (big endian) order; the uint32 represented by the hex bytes 01 02 03 04 is equivalent to the decimal value 16909060.

The protocol defines several different messages, each of which containing the the following information:

- o Protocol version
- o Message type identifier
- o The message body

Each of these messages is a "TunnelMessage" in the syntax, with a message type indicating the actual content of the message body.

6.1. Tunnel Message Format

The syntax of the protocol is defined below. "TunnelMessage" defines the structure of all messages sent via the tunnel protocol. That structure includes a field called "msg_type" that identifies the specific type of message contained within "TunnelMessage".

```
enum {
    unsupported_version(1),
    supported_profiles(2),
    media_keys(3),
    tunneled_dtls(4),
    endpoint_disconnect(5),
    (255)
} MsgType;

struct {
    uint8 version;
    MsgType msg_type;
    select (MsgType) {
        case unsupported_version: UnsupportedVersion;
        case supported_profiles:  SupportedProfiles;
        case media_keys:          MediaKeys;
        case tunneled_dtls:       TunneledDtls;
        case endpoint_disconnect: EndpointDisconnect;
    } body;
} TunnelMessage;
```

The elements of "TunnelMessage" include:

- o version: indicates the version of this protocol (0x00).
- o msg_type: the type of message contained within the structure "body".

The "UnsupportedVersion" message is defined as follows:

```
struct { } UnsupportedVersion;
```

The "UnsupportedVersion" message does not convey any additional information in the body.

The "SupportedProfiles" message is defined as:

```
uint8 SRTPProtectionProfile[2]; // from RFC5764
```

```
struct {  
    SRTPProtectionProfile protection_profiles<0..2^16-1>;  
} SupportedProfiles;
```

This message contains this single element: *protection_profiles: The list of two-octet SRTP protection profile values as per [RFC5764] supported by the media distributor.

The "MediaKeys" message is defined as:

```
struct {  
    uint32 association_id;  
    SRTPProtectionProfile protection_profile;  
    opaque mki<0..255>;  
    opaque client_write_SRTP_master_key<1..255>;  
    opaque server_write_SRTP_master_key<1..255>;  
    opaque client_write_SRTP_master_salt<1..255>;  
    opaque server_write_SRTP_master_salt<1..255>;  
    opaque conf_id<0..255>;  
} MediaKeys;
```

The fields are described as follows:

- o association_id: A value that identifies a distinct DTLS association between an endpoint and the key distributor.
- o protection_profiles: The value of the two-octet SRTP protection profile value as per [RFC5764] used for this DTLS association.
- o mki: Master key identifier [RFC3711].
- o client_write_SRTP_master_key: The value of the SRTP master key used by the client (endpoint).
- o server_write_SRTP_master_key: The value of the SRTP master key used by the server (media distributor).

- o `client_write_SRTP_master_salt`: The value of the SRTP master salt used by the client (endpoint).
- o `server_write_SRTP_master_salt`: The value of the SRTP master salt used by the server (media distributor).
- o `conf_id`: Identifier that uniquely specifies which conference the media distributor should place this media flow in.

The "TunneledDtls" message is defined as:

```
struct {  
    uint32 association_id;  
    opaque conf_id<0..255>;  
    opaque dtls_message<0..2^16-1>;  
} TunneledDtls;
```

The fields are described as follows:

- o `association_id`: An value that identifies a distinct DTLS association between an endpoint and the key distributor.
- o `conf_id`: Optional identifier that uniquely specifies which conference this media flow is in.
- o `dtls_message`: the content of the DTLS message received by the endpoint or to be sent to the endpoint.

The "EndpointDisconnect" message is defined as:

```
struct {  
    uint32 association_id;  
} EndpointDisconnect;
```

The fields are described as follows:

- o `association_id`: An value that identifies a distinct DTLS association between an endpoint and the key distributor.

7. Example Binary Encoding

The "TunnelMessage" is encoded in binary following the procedures specified in [!RFC5246]. This section provides an example of what the bits on the wire would look like for the "SupportedProfiles" message that advertises support for both SRTP_AEAD_AES_128_GCM and SRTP_AEAD_AES_256_GCM [RFC7714].

```

TunnelMessage:
    version: 0x00
    message_type: 0x01
    SupportedProfiles:
        protection_profiles: 0x0004 (length)
                             0x00070008 (value)

```

Thus, the encoding on the wire presented here in network bytes order would be this stream of octets:

```
0x0001000400070008
```

8. IANA Considerations

This document establishes a new registry to contain message type values used in the DTLS Tunnel protocol. These data type values are a single octet in length. This document defines the values shown in Table 1 below, leaving the balance of possible values reserved for future specifications:

MsgType	Description
0x01	Unsupported Version
0x02	Supported SRTP Protection Profiles
0x03	Media Keys
0x04	Tunneled DTLS
0x05	Endpoint Disconnect

Table 1: Data Type Values for the DTLS Tunnel Protocol

The value 0x00 and all values in the range 0x06 to 0xFF are reserved.

The name for this registry is "Datagram Transport Layer Security (DTLS) Tunnel Protocol Data Types for Privacy Enhanced Conferencing".

9. Security Considerations

The encapsulated data is protected by the TLS connection from the endpoint to key distributor, and the media distributor is merely an on path entity. The media distributor does not have access to the end-to-end keying material. This does not introduce any additional security concerns beyond a normal DTLS-SRTP association.

The HBH keying material is protected by the mutual authenticated TLS connection between the media distributor and key distributor. The key distributor MUST ensure that it only forms associations with

authorized media distributors or it could hand HBH keying material to untrusted parties.

The supported profiles information sent from the media distributor to the key distributor is not particularly sensitive as it only provides the cryptographic algorithms supported by the media distributor. Further, it is still protected by the TLS connection between the media distributor and the key distributor.

10. Acknowledgments

The author would like to thank David Benham and Cullen Jennings for reviewing this document and providing constructive comments.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<http://www.rfc-editor.org/info/rfc3550>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<http://www.rfc-editor.org/info/rfc3711>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<http://www.rfc-editor.org/info/rfc5764>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

11.2. Informative References

[RFC7714] McGrew, D. and K. Igoe, "AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)", RFC 7714, DOI 10.17487/RFC7714, December 2015, <<http://www.rfc-editor.org/info/rfc7714>>.

Authors' Addresses

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, North Carolina 27709
USA

Phone: +1 919 476 2048
Email: paulej@packetizer.com

Paul M. Ellenbogen
Princeton University

Phone: +1 206 851 2069
Email: pe5@cs.princeton.edu

Nils H. Ohlmeier
Mozilla

Phone: +1 408 659 6457
Email: nils@ohlmeier.org