Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

C. Jennings
P. Jones
R. Barnes
Cisco Systems
A. Roach
Mozilla

March 5, 2018

SRTP Double Encryption Procedures
draft-ietf-perc-double-08

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 operations to provide hop-by-hop and end-to-end security guarantees. Both the end-to-end and hop-by-hop cryptographic algorithms 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 https://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 6, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Cryptographic Context 4
3.1. Key Derivation 5
4. Original Header Block 5
5. RTP Operations 6
5.1. Encrypting a Packet 6
5.2. Relaying a Packet 7
5.3. Decrypting a Packet 8
6. RTCP Operations 9
7. Use with Other RTP Mechanisms 9
7.1. RTX 9
7.2. RED 10
7.3. FEC 10
7.4. DTMF 11
8. Recommended Inner and Outer Cryptographic Algorithms 11
9. Security Considerations 11
10. IANA Considerations 12
10.1. DTLS-SRTP 13
11. Acknowledgments 14
12. References 14
12.1. Normative References 14
12.2. Informative References 14
Appendix A. Encryption Overview 16
Authors' Addresses 16

1. Introduction

Cloud conferencing systems that are based on switched conferencing have a central Media Distributor device 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 key 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 Media Distributor. At the same time, a separate cryptographic key provides integrity and optional confidentiality for the media flowing between the Media Distributor and the endpoints.

The framework document [I-D.ietf-perc-private-media-framework] describes this concept in more detail.

This specification defines an SRTP transform that uses the AES-GCM algorithm [RFC7714] to provide encryption and integrity for an RTP packet for the end-to-end cryptographic key as well as a hop-by-hop cryptographic encryption and integrity between the endpoint and the Media Distributor. The Media Distributor decrypts and checks integrity of the hop-by-hop security. The Media Distributor 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 algorithm before it is sent. The receiving endpoint decrypts and checks integrity using the hop-by-hop cryptographic algorithm and then replaces any parameters the Media Distributor changed using the information in the Original Header Block before decrypting and checking the end-to-end integrity.

One can think of the double as a normal SRTP transform for encrypting the RTP in a way where things that only know half of the key, can decrypt and modify part of the RTP packet but not other parts of if including the media payload.

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:

- Media Distributor: media distribution device that routes media from one endpoint to other endpoints
- end-to-end: meaning the link from one endpoint through one or more Media Distributors to the endpoint at the other end.
- hop-by-hop: meaning the link from the endpoint to or from the Media Distributor.
- OHB: Original Header Block is an octet string that contains the original values from the RTP header that might have been changed by a Media Distributor.

3. Cryptographic Context

This specification uses a cryptographic context with two parts: an inner (end-to-end) part 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) part that is used between endpoints and Media Distributors to ensure the integrity of media over a single hop and to enable a Media Distributor to modify certain RTP header fields. RTCP is also handled using the hop-by-hop cryptographic part. The RECOMMENDED cipher for the hop-by-hop and end-to-end algorithm 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 algorithms are generated with the following steps:

- Generate key and salt values of the length required for the combined inner (end-to-end) and outer (hop-by-hop) algorithms.
- Assign the key and salt values generated for the inner (end-to-end) algorithm to the first half of the key and the first half of the salt for the double algorithm.
- Assign the key and salt values for the outer (hop-by-hop) algorithm to the second half of the key and second half of the salt for the double algorithm. The first half of the key is referred to as the inner key while the second half is referred to as the outer key. When a key is used by a cryptographic algorithm, the salt used is the part of the salt generated with that key.
- the SSRC is the same for both the inner and out outer algorithms as it can not be changed.
- The SEQ and ROC are tracked independently for the inner and outer algorithms.

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

3.1. Key Derivation

In order to allow the inner and outer keys to be managed independently via the master key, the transforms defined in this document MUST be used with the following PRF, which preserves the separation between the two halves of the key:

```
PRF_double_n(k_master,x) = PRF_inner_(n/2)(k_master,x) ||
                           PRF_outer_(n/2)(k_master,x)

PRF_inner_n(k_master,x)  = PRF_n(inner(k_master),x)
PRF_outer_n(k_master,x)  = PRF_n(outer(k_master),x)
```

Here "PRF_n(k, x)" represents the default SRTP PRF [RFC3711], "inner(key)" represents the first half of the key, and "outer(key)" represents the second half of the key.

4. Original Header Block

The Original Header Block (OHB) contains the original values of any modified header fields. In the encryption process, the OHB is appended to the RTP payload. In the decryption process, the receiving endpoint uses it to reconstruct the original RTP header, so that it can pass the proper AAD value to the inner transform.

The OHB can reflect modifications to the following fields in an RTP header: the payload type, the sequence number, and the marker bit. All other fields in the RTP header MUST remain unmodified; since the OHB cannot reflect their original values, the receiver will be unable to verify the E2E integrity of the packet.

The OHB has the following syntax (in ABNF):

```
BYTE = %x00-FF

PT = BYTE
SEQ = 2BYTE
Config = BYTE

OHB = ?PT ?SEQ Config
```

If present, the PT and SEQ parts of the OHB contain the original payload type and sequence number fields, respectively. The final "config" octet of the OHB specifies whether these fields are present, and the original value of the marker bit (if necessary):

```
+-+-+-+-+-+-+-+-+
|R R R R B M P Q|
+-+-+-+-+-+-+-+-+
```

- o P: PT is present
- o Q: SEQ is present
- o M: Marker bit is present
- o B: Value of marker bit
- o R: Reserved, MUST be set to 0

In particular, an all-zero OHB config octet (0x00) indicates that there have been no modifications from the original header.

5. RTP Operations

5.1. Encrypting a Packet

To encrypt a packet, the endpoint encrypts the packet using the inner (end-to-end) cryptographic key and then encrypts using the outer (hop-by-hop) cryptographic key. The encryption also supports a mode for repair packets that only does the outer (hop-by-hop) encryption. The processes is as follows:

1. Form an RTP packet. If there are any header extensions, they MUST use [RFC8285].
2. If the packet is for repair mode data, skip to step 6.
3. Form a synthetic RTP packet with the following contents:
    * Header: The RTP header of the original packet with the following modifications:
    * The X bit is set to zero
    * The header is truncated to remove any extensions (12 + 4 * CC bytes)
    * Payload: The RTP payload of the original packet
4. Apply the inner cryptographic algorithm to the synthetic RTP packet from the previous step.

5. Replace the header of the protected RTP packet with the header of the original packet, and append to the payload of the packet (1) the authentication tag from the original transform, and (2) an empty OHB (0x00).

6. Apply the outer cryptographic algorithm 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 key.

When using EKT [I-D.ietf-perc-srtp-ekt-diet], the EKT Field comes after the SRTP packet exactly like using EKT with any other SRTP transform.

5.2. Relaying a Packet

The Media Distributor has the part of the key for the outer (hop-by-hop), but it does not have the part of the key for the (end-to-end) cryptographic algorithm. The cryptographic algorithm 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 algorithm and key.

In order to modify a packet, the Media Distributor 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 using the outer (hop-by-hop) key.

1. Apply the outer (hop-by-hop) cryptographic algorithm to decrypt the packet. If decrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used. Note that the RTP payload produced by this decryption operation contains the original encrypted payload with the tag from the inner transform and the OHB appended.

2. Change any parts of the RTP packet that the relay wishes to change and should be changed.

3. A Media Distributor can add information to the OHB, but MUST NOT change existing information in the OHB. If RTP value is changed and not already in the OHB, then add it with its original value to the OHB.

4. If the Media Distributor resets a parameter to its original value, it MAY drop it from the OHB. Note that this might result in a decrease in the size of the OHB.

5. Apply the outer (hop-by-hop) cryptographic algorithm to the packet. If the RTP Sequence Number has been modified, SRTP processing happens as defined in SRTP and will end up using the

new Sequence Number. 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 (hop-by-hop) cryptographic key, then uses the OHB to reconstruct the original packet, which it decrypts and verifies with the inner (end-to-end) cryptographic key.

1. Apply the outer cryptographic algorithm 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 key.

2. If the packet is for repair mode data, skip the rest of the steps. Note that the packet that results from the repair algorithm will still have encrypted data that needs to be decrypted as specified by the repair algorithm sections.

3. Remove the inner authentication tag and the OHB from the end of the payload of the outer SRTP packet.

4. Form a new synthetic SRTP packet with:

   * Header = Received header, with the following modifications:

   * Header fields replaced with values from OHB (if any)

   * The X bit is set to zero

   * The header is truncated to remove any extensions (12 + 4 * CC bytes)

   * Payload is the encrypted payload from the outer SRTP packet (after the inner tag and OHB have been stripped).

   * Authentication tag is the inner authentication tag from the outer SRTP packet.

5. Apply the inner cryptographic algorithm to this synthetic SRTP packet. Note if the RTP Sequence Number was changed by the Media Distributor, the synthetic packet has the original Sequence Number. If the integrity check does not pass, discard the packet.

Once the packet has been 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:

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

The PT and sequence number from the inner SRTP packet can be used for collection of various statistics.

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

- Mixer-to-client audio level indicators (See [RFC6465])

6. RTCP Operations

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

7. Use with Other RTP Mechanisms

There are some RTP related extensions that need special consideration to be used by a relay when using the double transform due to the end-to-end protection of the RTP. The repair mechanism, when used with double, typically operate on the double encrypted data then take the results of theses operations and encrypted them using only the HBH key. This results in three cryptography operation happening to the repair data sent over the wire.

7.1. RTX

When using RTX [RFC4588] with double, the cached payloads MUST be the encrypted packets with the bits that are sent over the wire to the other side. When encrypting a retransmission packet, it MUST be encrypted in packet repair mode.

A typical RTX receiver would decrypt the packet, undo the RTX transformation, then process the resulting packet using the normally by using the steps in Section 5.3.

7.2. RED

When using RED [RFC2198] with double, the primary encoding MAY contain RTP header extensions and CSRC identifiers but non primary encodings can not.

The sender takes encrypted payloads from the cached packets to form the RED payload. Any header extensions from the primary encoding are copied to the RTP packet that will cary the RED payload and the other RTP header information such as SSRC, SEQ, CSRC, etc are set to the same as the primary payload. The RED RTP packet is then encrypted in repair mode and sent.

The receiver decrypts the payload to find the RED payload. Note a media relay can do this decryption as the packet was sent in repair mode that only needs the hop-by-hop key. The RTP headers and header extensions along with the primary payload and PT from inside the RED payload are used to form the encrypted primary RTP packet which can then be decrypted with double. The RTP headers (but not header extensions or CSRC) along with PT from inside the RED payload are used for from the non primary payloads. The time offset information in the RED data MUST be used to adjust the sequence number in the RTP header by using the timestamp offset and packet rate to find a sequence number offset to adjust by. At this point the non primary packets can be decrypted with double.

Note that Flex FEC [I-D.ietf-payload-flexible-fec-scheme] is a superset of the capabilities of RED. For most applications, FlexFEC is a better choice than RED.

7.3. FEC

When using Flex FEC [I-D.ietf-payload-flexible-fec-scheme] with double, the negotiation of double for the crypto is the out of band signaling that indicates that the repair packets MUST use the order of operations of SRTP followed by FEC when encrypting. This is to ensure that the original media is not revealed to the Media Distributor but at the same time allow the Media Distributor to repair media. When encrypting a packet that contains the Flex FEC data, which is already encrypted, it MUST be encrypted in repair mode packet.

The algorithm recommend in [I-D.ietf-rtcweb-fec] for repair of video is Flex FEC [I-D.ietf-payload-flexible-fec-scheme]. Note that for interoperability with WebRTC, [I-D.ietf-rtcweb-fec] recommends not using additional FEC only m-line in SDP for the repair packets.

7.4. DTMF

When DTMF is sent with [RFC4733], it is end-to-end encrypted and the relay can not read it so it can not be used to control the relay. Other out of band methods to control the relay need to be used instead.

8. Recommended Inner and Outer Cryptographic Algorithms

This specification recommends and defines AES-GCM as both the inner and outer cryptographic algorithms, identified as DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM and DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM. These algorithm provide for authenticated encryption and will consume additional processing time double-encrypting for hop-by-hop and end-to-end. 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 algorithm)_(outer algorithm).

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 crypto 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 algorithm. Similarly, if a new transform was defined that provided only integrity, that would also be reasonable to use for the hop-by-hop as the payload data is already encrypted by the end-to-end.

The AES-GCM cryptographic algorithm introduces an additional 16 octets to the length of the packet. When using AES-GCM for both the inner and outer cryptographic algorithms, 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. For packets in repair mode, the data they are caring is often already encrypted further increasing the size.

9. Security Considerations

To summarize what is encrypted and authenticated, we will refer to all the RTP fields except headers created by the sender and before the pay load as the initial envelope and the RTP payload information with the media as the payload. Any additional headers added by the sender or Media Distributor are referred to as the extra envelope.

The sender uses the end-to-end key to encrypts the payload and authenticate the payload + initial envelope which using an AEAD cipher results in a slight longer new payload. Then the sender uses the hop-by-hop key to encrypt the new payload and authenticate the initial envelope extra envelope and the new payload.

The Media Distributor has the hop-by-hop key so it can check the authentication of the received packet across the initial envelope, extra envelope and payload data but it can't decrypt the payload as it does not have the end-to-end key. It can add or change extra envelope information. It then authenticates the initial plus extra envelope information plus payload with a hop-by-hop key. This hop-by-hop for the outgoing packet is typically different than the hop-by-hop key for the incoming packet.

The receiver can check the authentication of the initial and extra envelope information from the Media Distributor. This, along with the OHB, is used to construct a synthetic packet that is should be identical initial envelope plus payload to one the sender created and the receiver can check that it is identical and then decrypt the original payload.

The end result is that if the authentications succeed, the receiver knows exactly what the payload and initial envelope the sender sent, as well as exactly which modifications were made by the Media Distributor and what extra envelope the Media Distributor send. The receive does not know exactly what extra envelope the sender sent.

It is obviously critical that the intermediary has only the outer (hop-by-hop) algorithm key and not the half of the key for the the inner (end-to-end) algorithm. 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 (end-to-end) and outer (hop-by-hop) key holders are the same as the security properties of classic SRTP.

10. IANA Considerations

10.1. DTLS-SRTP

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

Value	Profile	Reference
{0x00, 0x09}	DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM	RFCXXXX
{0x00, 0x0A}	DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM	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
```

The first half of the key and salt is used for the inner (end-to-end) algorithm and the second half is used for the outer (hop-by-hop) algorithm.

11. Acknowledgments

Thank you for reviews and improvements to this specification from Alex Gouaillard, David Benham, Magnus Westerlund, Nils Ohlmeier, Paul Jones, Roni Even, and Suhas Nandakumar. In addition, thank you to Sergio Garcia Murillo proposed the change of transporting the OHB information in the RTP payload instead of the RTP header.

12. References

12.1. 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, <https://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, <https://www.rfc-editor.org/info/rfc3711>.

[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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7714>.

[RFC8285] Singer, D., Desineni, H., and R. Even, Ed., "A General Mechanism for RTP Header Extensions", RFC 8285, DOI 10.17487/RFC8285, October 2017, <https://www.rfc-editor.org/info/rfc8285>.

12.2. Informative References

[I-D.ietf-payload-flexible-fec-scheme]
Singh, V., Begen, A., Zanaty, M., and G. Mandyam, "RTP Payload Format for Flexible Forward Error Correction (FEC)", draft-ietf-payload-flexible-fec-scheme-05 (work in progress), July 2017.

[I-D.ietf-perc-dtls-tunnel]
Jones, P., Ellenbogen, P., and N. Ohlmeier, "DTLS Tunnel between a Media Distributor and Key Distributor to Facilitate Key Exchange", draft-ietf-perc-dtls-tunnel-02 (work in progress), October 2017.

[I-D.ietf-perc-private-media-framework]
Jones, P., Benham, D., and C. Groves, "A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing", draft-ietf-perc-private-media-framework-05 (work in progress), October 2017.

[I-D.ietf-perc-srtp-ekt-diet]
Jennings, C., Mattsson, J., McGrew, D., Wing, D., and F. Andreasen, "Encrypted Key Transport for DTLS and Secure RTP", draft-ietf-perc-srtp-ekt-diet-06 (work in progress), October 2017.

[I-D.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-08 (work in progress), March 2018.

[RFC2198] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A., and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, DOI 10.17487/RFC2198, September 1997, <https://www.rfc-editor.org/info/rfc2198>.

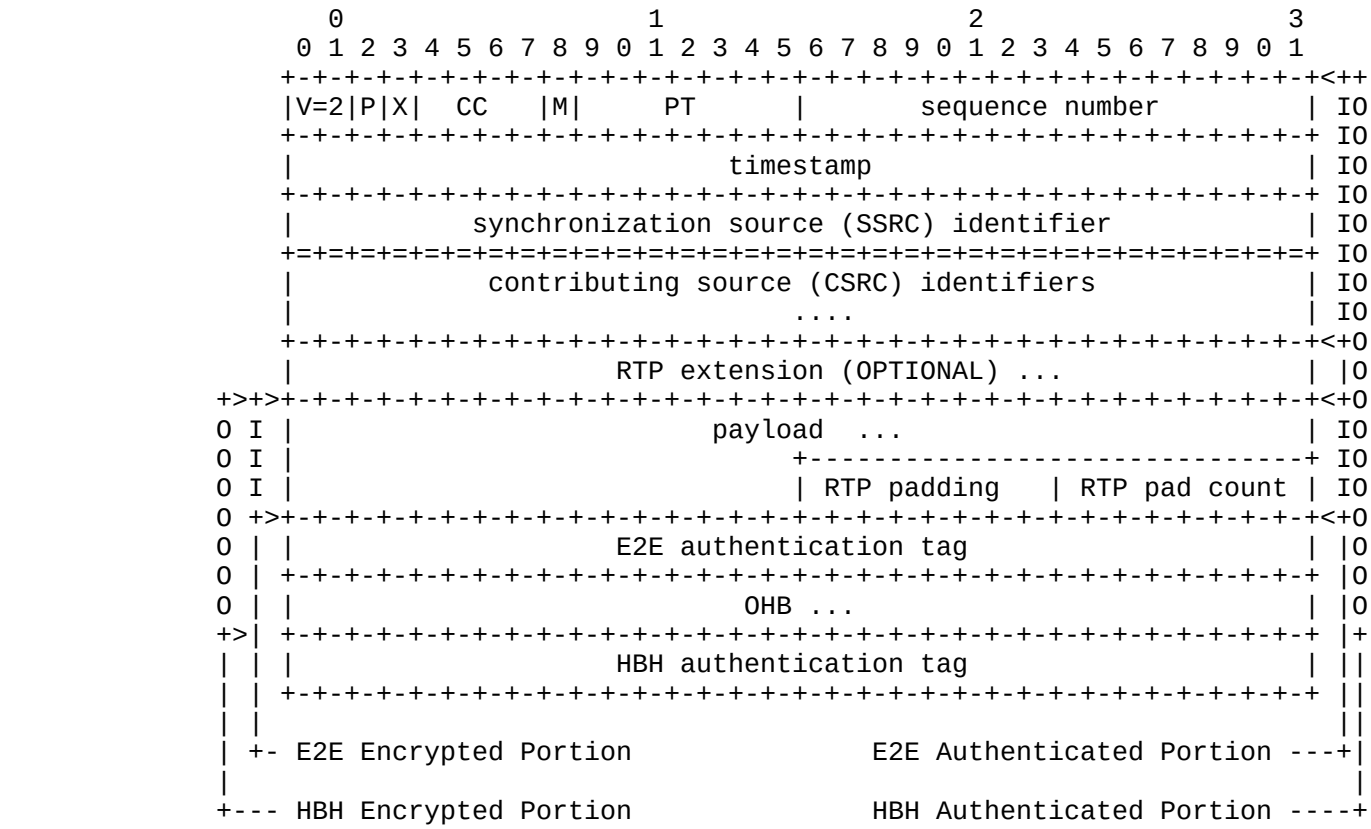
[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <https://www.rfc-editor.org/info/rfc4588>.

[RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals", RFC 4733, DOI 10.17487/RFC4733, December 2006, <https://www.rfc-editor.org/info/rfc4733>.

[RFC6465] Ivov, E., Ed., Marocco, E., Ed., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, DOI 10.17487/RFC6465, December 2011, <https://www.rfc-editor.org/info/rfc6465>.

Appendix A. Encryption Overview

The following figure shows a double encrypted SRTP packet. The sides indicate the parts of the packet that are encrypted and authenticated by the hob-by-hop and end-to-end operations.



Authors' Addresses

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca

Paul E. Jones
Cisco Systems

Email: paulej@packetizer.com

Richard Barnes
Cisco Systems

Email: rlb@ipv.sx

Adam Roach
Mozilla

Email: adam@nostrum.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 26, 2018

P. Jones
Cisco Systems
P. Ellenbogen
Princeton University
N. Ohlmeier
Mozilla

April 24, 2018

DTLS Tunnel between a Media Distributor and Key Distributor to Facilitate Key Exchange
draft-ietf-perc-dtls-tunnel-03

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 https://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 26, 2018.

Copyright Notice

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. Media Distributor Tunneling Procedures 7
5.4. Key Distributor Tunneling Procedures 8
5.5. Versioning Considerations 9
6. Tunneling Protocol 10
6.1. Tunnel Message Format 10
7. Example Binary Encoding 13
8. IANA Considerations 13
9. Security Considerations 14
10. Acknowledgments 14
11. References 14
11.1. Normative References 15
11.2. Informative References 16
Authors' Addresses 16

1. Introduction

An objective of Privacy-Enhanced RTP Conferencing (PERC) 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:

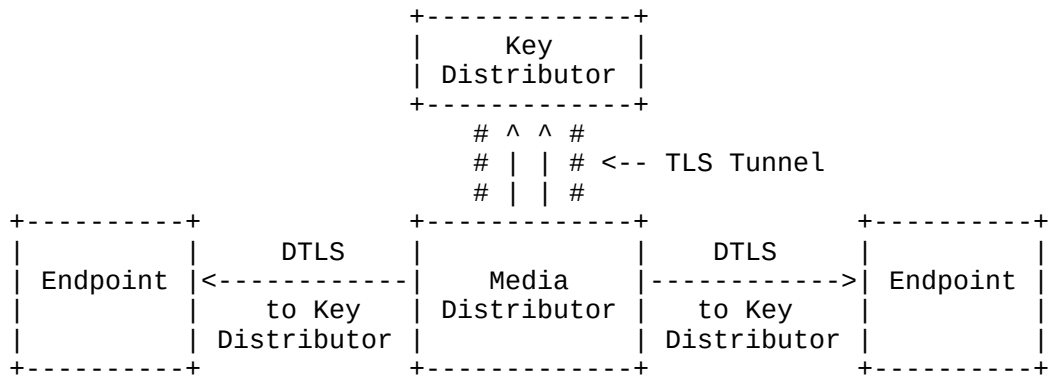


Figure 1: TLS Tunnel to Key Distributor

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 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.

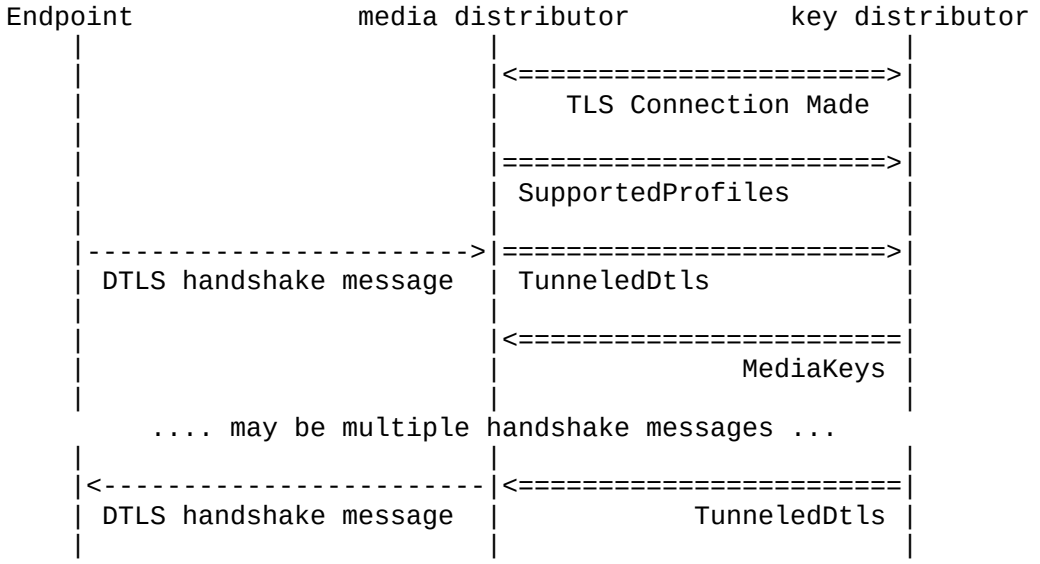


Figure 2: Sample DTLS-SRTP Exchange via the Tunnel

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 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.

The endpoint MUST include the "sdp_tls_id" DTLS extension [I-D.thomson-mmusic-sdp-uks] in the "ClientHello" message when establishing a DTLS association. Likewise, the "tls-id" SDP [RFC4566] attribute MUST be included in SDP sent by the endpoint in both the offer and answer [RFC3264] messages as per [I-D.ietf-mmusic-dtls-sdp].

When receiving a "tls_id" value from the key distributor, the client MUST check to ensure that value matches the "tls-id" value received in SDP. If the values do not match, the endpoint MUST consider any received keying material to be invalid and terminate the DTLS association.

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. 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 support the same list of protection profiles for the duration of any endpoint-initiated DTLS association and tunnel connection.

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 MUST be randomly assigned UUID [RFC4122] value.

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 "TuneledDtls" message. The media distributor is not required to 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.

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.

5.4. Key Distributor Tunneling Procedures

Each TLS tunnel established between the media distributor and the key distributor MUST be mutually authenticated.

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.

When processing an incoming endpoint association, the key distributor MUST extract the "tls_id" value transmitted in the "ClientHello" message and match that against "tls-id" value the endpoint transmitted via SDP. If the values in SDP and the "ClientHello" do not match, the DTLS association MUST be rejected.

The process through which the "tls-id" in SDP is conveyed to the key distributor is outside the scope of this document.

The key distributor MUST correlate the certificate fingerprint and "tls_id" received from endpoint’s "ClientHello" message with the corresponding values received from the SDP transmitted by the endpoint. It is through this correlation that the key distributor can be sure to deliver the correct conference key to the endpoint.

When sending the "ServerHello" message, the key distributor MUST insert its own "tls_id" value in the "sdp_tls_id" extension. This value MUST also be conveyed back to the client via SDP as a "tls-id" attribute.

The key distributor MUST encapsulate any DTLS message it sends to an endpoint inside a "TunneledDtls" message (see Section 6). The key distributor is not required to transmit all messages a given DTLS association through the same tunnel if more than one tunnel has been established between it and a media distributor.

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 uses the certificate fingerprint of the endpoint along with the "tls_id" value received in the "sdp_tls_id" extension to determine which conference a given DTLS association is associated.

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

When the DTLS association between the endpoint and the key distributor is terminated, regardless of which entity initiated the termination, the key distributor MUST send an "EndpointDisconnect" message with the association identifier assigned to the endpoint to the media distributor.

5.5. Versioning Considerations

All messages for an established tunnel MUST utilize the same version value.

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).

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:

- Protocol version
- Message type identifier
- 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 {
       supported_profiles(1),
       unsupported_version(2),
       media_keys(3),
       tunneled_dtls(4),
       endpoint_disconnect(5),
       (255)
   } MsgType;

   opaque uuid[16];

   struct {
       MsgType msg_type;
       uint16 length;
       select (MsgType) {
           case supported_profiles:  SupportedProfiles;
           case unsupported_version: UnsupportedVersion;
           case media_keys:          MediaKeys;
           case tunneled_dtls:       TunneledDtls;
           case endpoint_disconnect: EndpointDisconnect;
     } body;
   } TunnelMessage;
```

The elements of "TunnelMessage" include:

- msg_type: the type of message contained within the structure "body".
- length: the length in octets of the following "body" of the message.

The "SupportedProfiles" message is defined as:

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

struct {
  uint8 version;
  SRTPProtectionProfile protection_profiles<0..2^16-1>;
} SupportedProfiles;
```

This message contains this single element:

- version: indicates the version of this protocol (0x00).
- protection_profiles: The list of two-octet SRTP protection profile values as per [RFC5764] supported by the media distributor.

The "UnsupportedVersion" message is defined as follows:

```
struct {
    uint8 highest_version;
} UnsupportedVersion;
```

The elements of "UnsupportedVersion" include:

- highest_version: indicates the highest supported protocol version.

The "MediaKeys" message is defined as:

```
struct {
    uuid 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>;
} MediaKeys;
```

The fields are described as follows:

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

The "TunneledDtls" message is defined as:

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

The fields are described as follows:

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

- dtls_message: the content of the DTLS message received by the endpoint or to be sent to the endpoint.

The "EndpointDisconect" message is defined as:

```
struct {
    uuid association_id;
} EndpointDisconnect;
```

The fields are described as follows:

- 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 DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM and DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM [I-D.ietf-perc-double].

RFC Editor Note: Please replace the values 0009 and 000A in the following two examples with whatever code points IANA assigned for DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM and DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM.

```
TunnelMessage:
         message_type: 0x01
               length: 0x0007
    SupportedProfiles:
                  version:  0x00
      protection_profiles:  0x0004 (length)
                            0x0009000A (value)
```

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

```
0x0100070000040009000A
```

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	Supported SRTP Protection Profiles
0x02	Unsupported Version
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

[I-D.ietf-mmusic-dtls-sdp]
Holmberg, C. and R. Shpount, "Session Description Protocol (SDP) Offer/Answer Considerations for Datagram Transport Layer Security (DTLS) and Transport Layer Security (TLS)", draft-ietf-mmusic-dtls-sdp-32 (work in progress), October 2017.

[I-D.thomson-mmusic-sdp-uks]
Thomson, M. and E. Rescorla, "Unknown Key Share Attacks on uses of Transport Layer Security with the Session Description Protocol (SDP)", draft-thomson-mmusic-sdp-uks-00 (work in progress), April 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[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, <https://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, <https://www.rfc-editor.org/info/rfc3711>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6347>.

11.2. Informative References

[I-D.ietf-perc-double] Jennings, C., Jones, P., Barnes, R., and A. Roach, "SRTP Double Encryption Procedures", draft-ietf-perc-double-08 (work in progress), March 2018.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <https://www.rfc-editor.org/info/rfc4566>.

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

P. Jones
D. Benham
Cisco
C. Groves
Independent

March 5, 2018

A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing
draft-ietf-perc-private-media-framework-06

Abstract

This document describes a solution framework for ensuring that media confidentiality and integrity are maintained end-to-end within the context of a switched conferencing environment where media distributors are not trusted with the end-to-end media encryption keys. The solution aims to build upon existing security mechanisms defined for the real-time transport protocol (RTP).

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 https://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 6, 2018.

Table of Contents

1. Introduction 3
2. Conventions Used in This Document 4
3. PERC Entities and Trust Model 5
3.1. Untrusted Entities 5
3.1.1. Media Distributor 6
3.1.2. Call Processing 6
3.2. Trusted Entities 7
3.2.1. Endpoint 7
3.2.2. Key Distributor 7
4. Framework for PERC 7
4.1. End-to-End and Hop-by-Hop Authenticated Encryption . . . 8
4.2. E2E Key Confidentiality 9
4.3. E2E Keys and Endpoint Operations 9
4.4. HBH Keys and Hop Operations 10
4.5. Key Exchange 10
4.5.1. Initial Key Exchange and Key Distributor 11
4.5.2. Key Exchange during a Conference 12
5. Authentication 13
5.1. Identity Assertions 13
5.2. Certificate Fingerprints in Session Signaling 13
5.3. Conferences Identification 14
6. Security Considerations 14
6.1. Third Party Attacks 14
6.2. Media Distributor Attacks 15
6.2.1. Denial of service 15
6.2.2. Replay Attack 16
6.2.3. Delayed Playout Attack 16
6.2.4. Splicing Attack 16
7. IANA Considerations 16
8. Acknowledgments 17
9. References 17
9.1. Normative References 17
9.2. Informative References 18
Appendix A. PERC Key Inventory 19
A.1. DTLS-SRTP Exchange Yields HBH Keys 20
A.2. The Key Distributor Transmits the KEK (EKT Key) 20
A.3. Endpoints fabricate an SRTP Master Key 21
A.4. Who has What Key 21
Appendix B. PERC Packet Format 22
Authors' Addresses 23

1. Introduction

Switched conferencing is an increasingly popular model for multimedia conferences with multiple participants using a combination of audio, video, text, and other media types. With this model, real-time media flows from conference participants are not mixed, transcoded, transrated, recomposed, or otherwise manipulated by a Media Distributor, as might be the case with a traditional media server or multipoint control unit (MCU). Instead, media flows transmitted by conference participants are simply forwarded by the Media Distributor to each of the other participants, often forwarding only a subset of flows based on voice activity detection or other criteria. In some instances, the Media Distributors may make limited modifications to RTP [RFC3550] headers, for example, but the actual media content (e.g., voice or video data) is unaltered.

An advantage of switched conferencing is that Media Distributors can be more easily deployed on general-purpose computing hardware, including virtualized environments in private and public clouds. Deploying conference resources in a public cloud environment might introduce a higher security risk. Whereas traditional conference resources were usually deployed in private networks that were protected, cloud-based conference resources might be viewed as less secure since they are not always physically controlled by those who use them. Additionally, there are usually several ports open to the public in cloud deployments, such as for remote administration, and so on.

This document defines a solution framework wherein media privacy is ensured by making it impossible for a media distributor to gain access to keys needed to decrypt or authenticate the actual media content sent between conference participants. At the same time, the framework allows for the Media Distributors to modify certain RTP headers; add, remove, encrypt, or decrypt RTP header extensions; and encrypt and decrypt RTCP packets. The framework also prevents replay attacks by authenticating each packet transmitted between a given participant and the media distributor using a unique key per endpoint that is independent from the key for media encryption and authentication.

A goal of this document is to define a framework for enhanced privacy in RTP-based conferencing environments while utilizing existing security procedures defined for RTP with minimal enhancements.

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.

Additionally, this solution framework uses the following terms and acronyms:

End-to-End (E2E): Communications from one endpoint through one or more Media Distributors to the endpoint at the other end.

Hop-by-Hop (HBH): Communications between an endpoint and a Media Distributor or between Media Distributors.

Trusted Endpoint: An RTP flow terminating entity that has possession of E2E media encryption keys and terminates E2E encryption. This may include embedded user conferencing equipment or browsers on computers, media gateways, MCUs, media recording device and more that are in the trusted domain for a given deployment.

Media Distributor (MD): An RTP middlebox that is not allowed to to have access to E2E encryption keys. It operates according to the Selective Forwarding Middlebox RTP topologies [RFC7667] per the constraints defined by the PERC system, which includes, but not limited to, having no access to RTP media unencrypted and having limits on what RTP header field it can alter.

Key Distributor: An entity that is a logical function which distributes keying material and related information to trusted endpoints and Media Distributor(s), only that which is appropriate for each. The Key Distributor might be co-resident with another entity trusted with E2E keying material.

Conference: Two or more participants communicating via trusted endpoints to exchange RTP flows through one or more Media Distributor.

Call Processing: All trusted endpoints in the conference connect to it by a call processing dialog, such as with the Focus defined in the Framework for Conferencing with SIP [RFC4353].

Third Party: Any entity that is not an Endpoint, Media Distributor, Key Distributor or Call Processing entity as described in this document.

3. PERC Entities and Trust Model

The following figure depicts the trust relationships, direct or indirect, between entities described in the subsequent sub-sections. Note that these entities may be co-located or further divided into multiple, separate physical devices.

Please note that some entities classified as untrusted in the simple, general deployment scenario used most commonly in this document might be considered trusted in other deployments. This document does not preclude such scenarios, but will keep the definitions and examples focused by only using the the simple, most general deployment scenario.

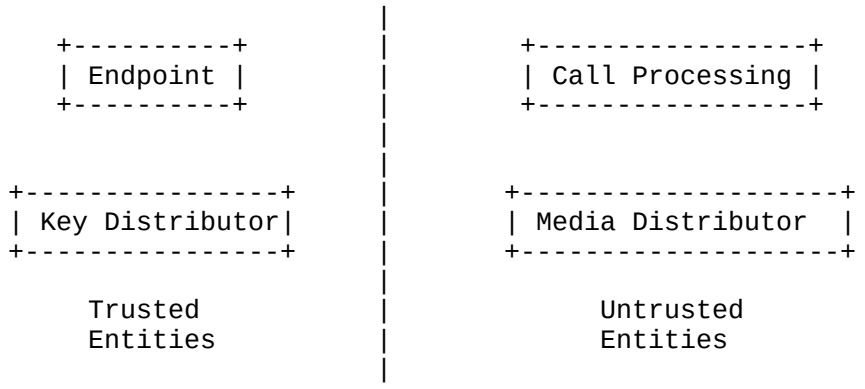


Figure 1: Trusted and Untrusted Entities in PERC

3.1. Untrusted Entities

The architecture described in this framework document enables conferencing infrastructure to be hosted in domains, such as in a cloud conferencing provider's facilities, where the trustworthiness is below the level needed to assume the privacy of participant's media will not be compromised. The conferencing infrastructure in such a domain is still trusted with reliably connecting the participants together in a conference, but not trusted with keying material needed to decrypt any of the participant's media. Entities in such lower trustworthiness domains will simply be referred to as untrusted entities from this point forward.

It is important to understand that untrusted in this document does not mean an entity is not expected to function properly. Rather, it

means only that the entity does not have access to the E2E media encryption keys.

3.1.1. Media Distributor

A Media Distributor forwards RTP flows between endpoints in the conference while performing per-hop authentication of each RTP packet. The Media Distributor may need access to one or more RTP headers or header extensions, potentially adding or modifying a certain subset. The Media Distributor will also relay secured messaging between the endpoints and the Key Distributor and will acquire per-hop key information from the Key Distributor. The actual media content MUST NOT not be decryptable by a Media Distributor, so it is untrusted to have access to the E2E media encryption keys. The key exchange mechanisms specified in this framework will prevent the Media Distributor from gaining access to the E2E media encryption keys.

An endpoint’s ability to join a conference hosted by a Media Distributor MUST NOT alone be interpreted as being authorized to have access to the E2E media encryption keys, as the Media Distributor does not have the ability to determine whether an endpoint is authorized. Instead, the Key Distributor is responsible for authenticating endpoints (e.g., using WebRTC Identity [I-D.ietf-rtcweb-security-arch]) and determining their authorization to receive E2E media encryption keys.

A Media Distributor MUST perform its role in properly forwarding media packets while taking measures to mitigate the adverse effects of denial of service attacks (refer to Section 6), etc, to a level equal to or better than traditional conferencing (i.e. non-PERC) deployments.

A Media Distributor or associated conferencing infrastructure may also initiate or terminate various conference control related messaging, which is outside the scope of this framework document.

3.1.2. Call Processing

The call processing function is untrusted in the simple, general deployment scenario. When a physical subset of the call processing function resides in facilities outside the trusted domain, it should not be trusted to have access to E2E key information.

The call processing function may include the processing of call signaling messages, as well as the signing of those messages. It may also authenticate the endpoints for the purpose of call signaling and subsequently joining of a conference hosted through one or more Media

Distributors. Call processing may optionally ensure the privacy of call signaling messages between itself, the endpoint, and other entities.

In any deployment scenario where the call processing function is considered trusted, the call processing function MUST ensure the integrity of received messages before forwarding to other entities.

3.2. Trusted Entities

From the PERC model system perspective, entities considered trusted (refer to Figure 1) can be in possession of the E2E media encryption keys for one or more conferences.

3.2.1. Endpoint

An endpoint is considered trusted and will have access to E2E key information. While it is possible for an endpoint to be compromised, subsequently performing in undesired ways, defining endpoint resistance to compromise is outside the scope of this document. Endpoints will take measures to mitigate the adverse effects of denial of service attacks (refer to Section 6) from other entities, including from other endpoints, to a level equal to or better than traditional conference (i.e., non-PERC) deployments.

3.2.2. Key Distributor

The Key Distributor, which may be colocated with an endpoint or exist standalone, is responsible for providing key information to endpoints for both end-to-end and hop-by-hop security and for providing key information to Media Distributors for the hop-by-hop security.

Interaction between the Key Distributor and the call processing function is necessary to for proper conference-to-endpoint mappings. This is described in Section 5.3.

The Key Distributor needs to be secured and managed in a way to prevent exploitation by an adversary, as any kind of compromise of the Key Distributor puts the security of the conference at risk.

4. Framework for PERC

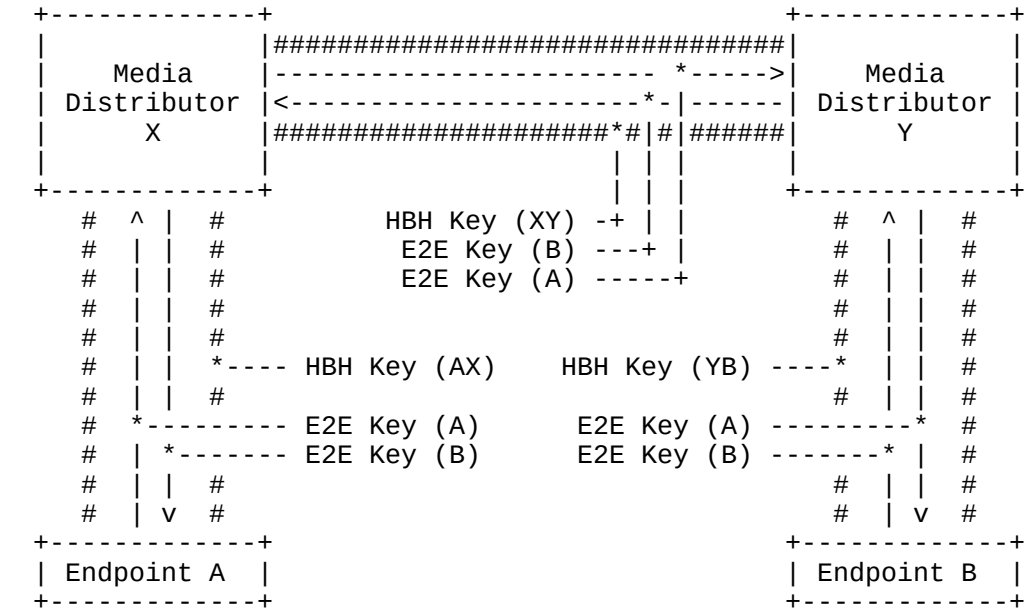
The purpose for this framework is to define a means through which media privacy can be ensured when communicating within a conferencing environment consisting of one or more Media Distributors that only switch, hence not terminate, media. It does not otherwise attempt to hide the fact that a conference between endpoints is taking place.

This framework reuses several specified RTP security technologies, including SRTP [RFC3711], PERC EKT [I-D.ietf-perc-srtp-ekt-diet], and DTLS-SRTP [RFC5764].

4.1. End-to-End and Hop-by-Hop Authenticated Encryption

This solution framework focuses on the end-to-end privacy and integrity of the participant's media by limiting access of the end-to-end key information to trusted entities. However, this framework does give a Media Distributor access to RTP headers and all or most header extensions, as well as the ability to modify a certain subset of those headers and to add header extensions. Packets received by a Media Distributor or an endpoint are authenticated hop-by-hop.

To enable all of the above, this framework defines the use of two security contexts and two associated encryption keys: an "inner" key (an E2E key distinct for each transmitted media flow) for authenticated encryption of RTP media between endpoints and an "outer" key (HBH key) known only to media distributor and the adjacent endpoint) for the hop between an endpoint and a Media Distributor or between Media Distributor.



E2E and HBH Keys Used for Authenticated Encryption of SRTP Packets

The PERC Double transform [I-D.ietf-perc-double] enables endpoints to perform encryption using both the E2E and HBH contexts while still

preserving the same overall interface as other SRTP transforms. The Media Distributor simply uses the corresponding normal (single) AES-GCM transform, keyed with the appropriate HBH keys. See Appendix A for a description of the keys used in PERC and Appendix B for an overview of how the packet appears on the wire.

RTCP can only be encrypted hop-by-hop, not end-to-end. This framework introduces no additional step for RTCP authenticated encryption, so the procedures needed are specified in [RFC3711] and use the same outer, hop-by-hop cryptographic context chosen in the Double operation described above.

4.2. E2E Key Confidentiality

To ensure the confidentiality of E2E keys shared between endpoints, endpoints will make use of a common Key Encryption Key (KEK) that is known only by the trusted entities in a conference. That KEK, defined in the PERC EKT [I-D.ietf-perc-srtp-ekt-diet] as the EKT Key, will be used to subsequently encrypt the SRTP master key used for E2E authenticated encryption of media sent by a given endpoint. Each endpoint in the conference will create a random SRTP master key for E2E authenticated encryption, thus participants in the conference MUST keep track of the E2E keys received via the Full EKT Field for each distinct SSRC in the conference so that it can properly decrypt received media. Note, too, that an endpoint may change its E2E key at any time and advertise that new key to the conference as specified in [I-D.ietf-perc-srtp-ekt-diet].

4.3. E2E Keys and Endpoint Operations

Any given RTP media flow can be identified by its SSRC, and endpoints might send more than one at a time and change the mix of media flows transmitted during the life of a conference.

Thus, endpoints MUST maintain a list of SSRCs from received RTP flows and each SSRC’s associated E2E key information. Following a change in an E2E key, prior E2E keys SHOULD be retained by receivers for a period long enough to ensure that late-arriving or out-of-order packets from the endpoint can be successfully decrypted. Receiving endpoints MUST discard old E2E keys no later than when it leaves the conference.

If there is a need to encrypt one or more RTP header extensions end-to-end, an encryption key is derived from the end-to-end SRTP master key to encrypt header extensions as per [RFC6904]. The Media Distributor will not be able use the information contained in those header extensions encrypted with an E2E key.

4.4. HBH Keys and Hop Operations

To ensure the integrity of transmitted media packets, this framework requires that every packet be authenticated hop-by-hop (HBH) between an endpoint and a Media Distributor, as well between Media Distributors. The authentication key used for hop-by-hop authentication is derived from an SRTP master key shared only on the respective hop. Each HBH key is distinct per hop and no two hops ever use the same SRTP master key.

Using hop-by-hop authentication gives the Media Distributor the ability to change certain RTP header values. Which values the Media Distributor can change in the RTP header are defined in [I-D.ietf-perc-double]. RTCP can only be encrypted HBH, giving the Media Distributor the flexibility to forward RTCP content unchanged, transmit compound RTCP packets or to initiate RTCP packets for reporting statistics or conveying other information. Performing hop-by-hop authentication for all RTP and RTCP packets also helps provide replay protection (see Section 6).

If there is a need to encrypt one or more RTP header extensions hop-by-hop, an encryption key is derived from the hop-by-hop SRTP master key to encrypt header extensions as per [RFC6904]. This will still give the Media Distributor visibility into header extensions, such as the one used to determine audio level [RFC6464] of conference participants. Note that when RTP header extensions are encrypted, all hops - in the untrusted domain at least - will need to decrypt and re-encrypt these encrypted header extensions.

4.5. Key Exchange

In brief, the keys used by any given endpoints are determined in the following way:

- The HBH keys that the endpoint uses to send and receive SRTP media are derived from a DTLS handshake that the endpoint performs with the Key Distributor (following normal DTLS-SRTP procedures).

- The E2E key that an endpoint uses to send SRTP media can either be set from DTLS or chosen by the endpoint. It is then distributed to other endpoints in a Full EKT Field, encrypted under an EKTKey provided to the client by the Key Distributor within the DTLS channel they negotiated.

- Each E2E key that an endpoint uses to receive SRTP media is set by receiving a Full EKT Field from another endpoint.

4.5.1. Initial Key Exchange and Key Distributor

The Media Distributor maintains a tunnel with the Key Distrubutor (e.g., using [I-D.ietf-perc-dtls-tunnel]), making it possible for the Media Distributor to facilitate the establishment of a secure DTLS association between each endpoint and the Key Distributor as shown the following figure. The DTLS association between endpoints and the Key Distributor will enable each endpoint to generate E2E and HBH keys and receive the Key Encryption Key (KEK) (i.e., EKT Key). At the same time, the Key Distributor can securely provide the HBH key information to the Media Distributor. The key information summarized here may include the SRTP master key, SRTP master salt, and the negotiated cryptographic transform.

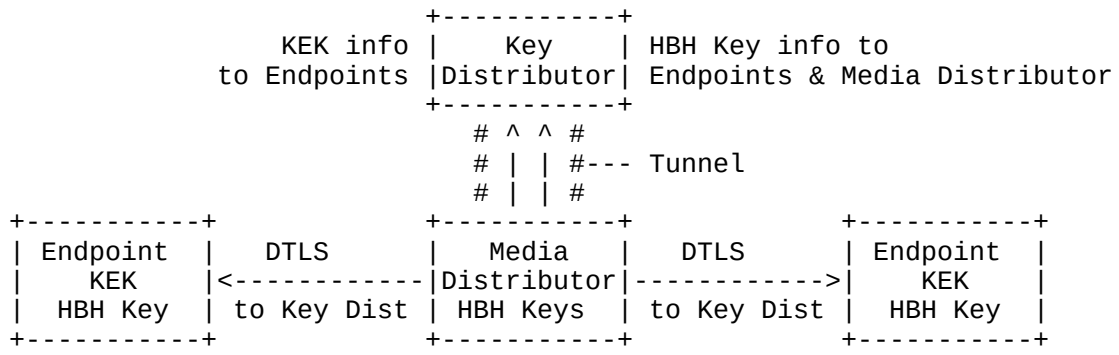


Figure 2: Exchanging Key Information Between Entities

Endpoints will establish a DTLS-SRTP [RFC5764] association over the RTP session's media ports for the purposes of key information exchange with the Key Distributor. The Media Distributor will not terminate the DTLS signaling, but will instead forward DTLS packets received from an endpoint on to the Key Distributor (and vice versa) via a tunnel established between Media Distributor and the Key Distributor. This tunnel is used to encapsulate the DTLS-SRTP signaling between the Key Distributor and endpoints will also be used to convey HBH key information from the Key Distributor to the Media Distributor, so no additional protocol or interface is required.

In establishing the DTLS association between endpoints and the Key Distributor, the endpoint MUST act as the DTLS client and the Key Distributor MUST act as the DTLS server. The Key Encryption Key (KEK) (i.e., EKT Key) is conveyed by the Key Distributor over the DTLS association to endpoints via procedures defined in PERC EKT [I-D.ietf-perc-srtp-ekt-diet] via the EKTKey message.

Note that following DTLS-SRTP procedures for the [I-D.ietf-perc-double] cipher, the endpoint will generate both E2E and HBH encryption keys and salt values. Endpoints MAY use the DTLS-SRTP generated E2E key for transmission or MAY generate a fresh E2E key. In either case, the generated SRTP master salt for E2E encryption MUST be replaced with the salt value provided by the Key Distributor via the EKTKey message. That is because every endpoint in the conference uses the same SRTP master salt. The endpoint only transmits the SRTP master key (not the salt) used for E2E encryption to other endpoints in RTP/RTCP packets per [I-D.ietf-perc-srtp-ekt-diet].

Media Distributors use DTLS-SRTP [RFC5764] directly with a peer Media Distributor to establish the HBH key for transmitting RTP and RTCP packets to that peer Media Distributor. The Key Distributor does not facilitate establishing a HBH key for use between Media Distributors.

4.5.2. Key Exchange during a Conference

Following the initial key information exchange with the Key Distributor, an endpoint will be able to encrypt media end-to-end with an E2E key, sending that E2E key to other endpoints encrypted with the KEK, and will be able to encrypt and authenticate RTP packets using a HBH key. The procedures defined do not allow the Media Distributor to gain access to the KEK information, preventing it from gaining access to any endpoint’s E2E key and subsequently decrypting media.

The KEK (i.e., EKT Key) may need to change from time-to-time during the life of a conference, such as when a new participant joins or leaves a conference. Dictating if, when or how often a conference is to be re-keyed is outside the scope of this document, but this framework does accommodate re-keying during the life of a conference.

When a Key Distributor decides to re-key a conference, it transmits a specific message defined in PERC EKT [I-D.ietf-perc-srtp-ekt-diet] to each of the conference participants. The endpoint MUST create a new SRTP master key and prepare to send that key inside a Full EKT Field using the new EKTKey. Since it may take some time for all of the endpoints in conference to finish re-keying, senders MUST delay a short period of time before sending media encrypted with the new master key, but it MUST be prepared to make use of the information from a new inbound EKT Key immediately. See Section 2.2.2 of [I-D.ietf-perc-srtp-ekt-diet].

Endpoints MAY follow the procedures in section 5.2 of [RFC5764] to re-negotiate HBH keys as desired. If new HBH keys are generated, the

new keys are also delivered to the Media Distributor following the procedures defined in [I-D.ietf-perc-dtls-tunnel].

Endpoints are at liberty to change the E2E encryption key used at any time. Endpoints MUST generate a new E2E encryption key whenever it receives a new EKT Key. After switching to a new key, the new key will be conveyed to other endpoints in the conference in RTP/RTCP packets per [I-D.ietf-perc-srtp-ekt-diet].

5. Authentication

It is important to this solution framework that the entities can validate the authenticity of other entities, especially the Key Distributor and endpoints. The details of this are outside the scope of specification but a few possibilities are discussed in the following sections. The key requirements is that endpoints can verify they are connected to the correct Key Distributor for the conference and the Key Distributor can verify the endpoints are the correct endpoints for the conference.

Two possible approaches to solve this are Identity Assertions and Certificate Fingerprints.

5.1. Identity Assertions

WebRTC Identity assertion [I-D.ietf-rtcweb-security-arch] can be used to bind the identity of the user of the endpoint to the fingerprint of the DTLS-SRTP certificate used for the call. This certificate is unique for a given call and a conference. This allows the Key Distributor to ensure that only authorized users participate in the conference. Similarly the Key Distributor can create a WebRTC Identity assertion to bind the fingerprint of the unique certificate used by the Key Distributor for this conference so that the endpoint can validate it is talking to the correct Key Distributor. Such a setup requires an Identity Provider (Idp) trusted by the endpoints and the Key Distributor.

5.2. Certificate Fingerprints in Session Signaling

Entities managing session signaling are generally assumed to be untrusted in the PERC framework. However, there are some deployment scenarios where parts of the session signaling may be assumed trustworthy for the purposes of exchanging, in a manner that can be authenticated, the fingerprint of an entity’s certificate.

As a concrete example, SIP [RFC3261] and SDP [RFC4566] can be used to convey the fingerprint information per [RFC5763]. An endpoint’s SIP User Agent would send an INVITE message containing SDP for the media

session along with the endpoint's certificate fingerprint, which can be signed using the procedures described in [RFC4474] for the benefit of forwarding the message to other entities by the Focus [RFC4353]. Other entities can now verify the fingerprints match the certificates found in the DTLS-SRTP connections to find the identity of the far end of the DTLS-SRTP connection and check that is the authorized entity.

Ultimately, if using session signaling, an endpoint's certificate fingerprint would need to be securely mapped to a user and conveyed to the Key Distributor so that it can check that that user is authorized. Similarly, the Key Distributor's certificate fingerprint can be conveyed to endpoint in a manner that can be authenticated as being an authorized Key Distributor for this conference.

5.3. Conferences Identification

The Key Distributor needs to know what endpoints are being added to a given conference. Thus, the Key Distributor and the Media Distributor will need to know endpoint-to-conference mappings, which is enabled by exchanging a conference-specific unique identifier as defined in [I-D.ietf-perc-dtls-tunnel]. How this unique identifier is assigned is outside the scope of this document.

6. Security Considerations

This framework, and the individual protocols defined to support it, must take care to not increase the exposure to Denial of Service (DoS) attacks by untrusted or third-party entities and should take measures to mitigate, where possible, more serious DoS attacks from on-path and off-path attackers.

The following section enumerates the kind of attacks that will be considered in the development of this framework's solution.

6.1. Third Party Attacks

On-path attacks are mitigated by HBH integrity protection and encryption. The integrity protection mitigates packet modification and encryption makes selective blocking of packets harder, but not impossible.

Off-path attackers may try connecting to different PERC entities and send specifically crafted packets. A successful attacker might be able to get the Media Distributor to forward such packets. If not making use of HBH authentication on the Media Distributor, such an attack could only be detected in the receiving endpoints where the forged packets would finally be dropped.

Another potential attack is a third party claiming to be a Media Distributor, fooling endpoints in to sending packets to the false Media Distributor instead of the correct one. The deceived sending endpoints could incorrectly assuming their packets have been delivered to endpoints when they in fact have not. Further, the false Media Distributor may cascade to another legitimate Media Distributor creating a false version of the real conference.

This attack can be mitigated by the false Media Distributor not being authenticated by the Key Distributor during PERC Tunnel establishment. Without the tunnel in place, endpoints will not establish secure associations with the Key Distributor and receive the KEK, causing the conference to not proceed.

6.2. Media Distributor Attacks

The Media Distributor can attack the session in a number of possible ways.

6.2.1. Denial of service

Any modification of the end-to-end authenticated data will result in the receiving endpoint getting an integrity failure when performing authentication on the received packet.

The Media Distributor can also attempt to perform resource consumption attacks on the receiving endpoint. One such attack would be to insert random SSRC/CSRC values in any RTP packet with an inband key-distribution message attached (i.e., Full EKT Field). Since such a message would trigger the receiver to form a new cryptographic context, the Media Distributor can attempt to consume the receiving endpoints resources.

Another denial of service attack is where the Media Distributor rewrites the PT field to indicate a different codec. The effect of this attack is that any payload packetized and encoded according to one RTP payload format is then processed using another payload format and codec. Assuming that the implementation is robust to random input, it is unlikely to cause crashes in the receiving software/ hardware. However, it is not unlikely that such rewriting will cause severe media degradation.

For audio formats, this attack is likely to cause highly disturbing audio and/or can be damaging to hearing and playout equipment.

6.2.2. Replay Attack

Replay attack is when an already received packets from a previous point in the RTP stream is replayed as new packet. This could, for example, allow a Media Distributor to transmit a sequence of packets identified as a user saying "yes", instead of the "no" the user actually said.

The mitigation for a replay attack is to prevent old packets beyond a small-to-modest jitter and network re-ordering sized window to be rejected. End-to-end replay protection MUST be provided for the whole duration of the conference.

6.2.3. Delayed Playout Attack

The delayed playout attack is a variant of the replay attack. This attack is possible even if E2E replay protection is in place. However, due to fact that the Media Distributor is allowed to select a sub-set of streams and not forward the rest to a receiver, such as in forwarding only the most active speakers, the receiver has to accept gaps in the E2E packet sequence. The issue with this is that a Media Distributor can select to not deliver a particular stream for a while.

Within the window from last packet forwarded to the receiver and the latest received by the Media Distributor, the Media Distributor can select an arbitrary starting point when resuming forwarding packets. Thus what the media source said can be substantially delayed at the receiver with the receiver believing that it is what was said just now, and only delayed due to transport delay.

6.2.4. Splicing Attack

The splicing attack is an attack where a Media Distributor receiving multiple media sources splices one media stream into the other. If the Media Distributor is able to change the SSRC without the receiver having any method for verifying the original source ID, then the Media Distributor could first deliver stream A and then later forward stream B under the same SSRC as stream A was previously using. Not allowing the Media Distributor to change the SSRC mitigates this attack.

7. IANA Considerations

There are no IANA considerations for this document.

8. Acknowledgments

The authors would like to thank Mo Zanaty and Christian Oien for invaluable input on this document. Also, we would like to acknowledge Nermeen Ismail for serving on the initial versions of this document as a co-author.

9. References

9.1. Normative References

[I-D.ietf-perc-double]
Jennings, C., Jones, P., Barnes, R., and A. Roach, "SRTP Double Encryption Procedures", draft-ietf-perc-double-08 (work in progress), March 2018.

[I-D.ietf-perc-dtls-tunnel]
Jones, P., Ellenbogen, P., and N. Ohlmeier, "DTLS Tunnel between a Media Distributor and Key Distributor to Facilitate Key Exchange", draft-ietf-perc-dtls-tunnel-02 (work in progress), October 2017.

[I-D.ietf-perc-srtp-ekt-diet]
Jennings, C., Mattsson, J., McGrew, D., Wing, D., and F. Andreasen, "Encrypted Key Transport for DTLS and Secure RTP", draft-ietf-perc-srtp-ekt-diet-07 (work in progress), March 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc3711>.

[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, <https://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, <https://www.rfc-editor.org/info/rfc6904>.

9.2. Informative References

[I-D.ietf-rtcweb-security-arch] Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-13 (work in progress), October 2017.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, DOI 10.17487/RFC4353, February 2006, <https://www.rfc-editor.org/info/rfc4353>.

[RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, DOI 10.17487/RFC4474, August 2006, <https://www.rfc-editor.org/info/rfc4474>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <https://www.rfc-editor.org/info/rfc4566>.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <https://www.rfc-editor.org/info/rfc5763>.

[RFC6464] Lennox, J., Ed., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <https://www.rfc-editor.org/info/rfc6464>.

[RFC7667] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 7667, DOI 10.17487/RFC7667, November 2015, <https://www.rfc-editor.org/info/rfc7667>.

Appendix A. PERC Key Inventory

PERC specifies the use of a number of different keys and, understandably, it looks complicated or confusing on the surface. This section summarizes the various keys used in the system, how they are generated, and what purpose they serve.

The keys are described in the order in which they would typically be acquired.

The various keys used in PERC are shown in Figure 3 below.

Key	Description
KEK (EKT Key)	Key shared by all endpoints and used to encrypt each endpoint's SRTP master key so receiving endpoints can decrypt media.
HBH Key	Key used to encrypt media hop-by-hop.
E2E Key	Key used to encrypt media end-to-end.

Figure 3: Key Inventory

As you can see, the number key types is very small. However, what can be challenging is keeping track of all of the distinct E2E keys as the conference grows in size. With 1,000 participants in a conference, there will be 1,000 distinct SRTP master keys, all of which share the same master salt. Each of those keys are passed through the KDF defined in [RFC3711] to produce the actual encryption and authentication keys. Complicating key management is the fact that the KEK can change and, when it does, the endpoints generate new SRTP master keys. And, of course, there is a new SRTP master salt to go with those keys. Endpoints have to retain old keys for a period of time to ensure they can properly decrypt late-arriving or out-of-order packets.

The time required to retain old keys (either EKT Keys or SRTP master keys) is not specified, but they should be retained at least for the period of time required to re-key the conference or handle late-arriving or out-of-order packets. A period of 60s should be considered a generous retention period, but endpoints may keep old keys on hand until the end of the conference.

Or more detailed explanation of each of the keys follows.

A.1. DTLS-SRTP Exchange Yields HBH Keys

The first set of keys acquired are for hop-by-hop encryption and decryption. Assuming the use of Double [I-D.ietf-perc-double], the endpoint would perform DTLS-SRTP exchange with the key distributor and receive a key that is, in fact, "double" the size that is needed. Per the Double specification, the E2E part is the first half of the key, so the endpoint will just discard that information in PERC. It is not used. The second half of the key material is for HBH operations, so that half of the key (corresponding to the least significant bits) is assigned internally as the HBH key.

The media distributor doesn't perform DTLS-SRTP, but it is at this point that the key distributor will inform the media distributor of the HBH key value via the tunnel protocol ([I-D.ietf-perc-dtls-tunnel]). The key distributor will send the least significant bits corresponding to the half of the keying material determined through DTLS-SRTP with the endpoint to the media distributor via the tunnel protocol. There is a salt generated along with the HBH key. The salt is also longer than needed for HBH operations, thus only the least significant bits of the required length (i.e., half of the generated salt material) are sent to the media distributor via the tunnel protocol.

No two endpoints will have the same HBH key, thus the media distributor must keep track each distinct HBH key (and the corresponding salt) and use it only for the specified hop.

This key is also used for HBH encryption of RTCP. RTCP is not end-to-end encrypted in PERC.

A.2. The Key Distributor Transmits the KEK (EKT Key)

Via the aforementioned DTLS-SRTP association, the key distributor will send the endpoint the KEK (i.e., EKT Key per [I-D.ietf-perc-srtp-ekt-diet]). This key is known only to the key distributor and endpoints. This key is the most important to protect since having knowledge of this key (and the SRTP master salt transmitted as a part of the same message) will allow an entity to decrypt any media packet in the conference.

Note that the key distributor can send any number of EKT Keys to endpoints. This can be used to re-key the entire conference. Each key is identified by a "Security Parameter Index" (SPI) value. Endpoints should expect that a conference might be re-keyed when a new participant joins a conference or when a participant leaves a conference in order to protect the confidentiality of the conversation before and after such events.

The SRTP master salt to be used by the endpoint is transmitted along with the EKT Key. All endpoints in the conference utilize the same SRTP master salt that corresponds with a given EKT Key.

The EKT Field in media packets is encrypted using a cipher specified via the EKTKey message (e.g., AES Key Wrap with a 128-bit key). This cipher is different than the cipher used to protect media and is only used to encrypt the endpoint's SRTP master key (and other EKT Field data as per [I-D.ietf-perc-srtp-ekt-diet]).

The media distributor is not given the KEK (i.e., EKT Key).

A.3. Endpoints fabricate an SRTP Master Key

As stated earlier, the E2E key determined via DTLS-SRTP MAY be discarded in favor of a locally-generated SRTP master key. While the DTLS-SRTP-derived key could be used, the fact that an endpoint might need to change the SRTP master key periodically or is forced to change the SRTP master key as a result of the EKT key changing means using it has only limited utility. To reduce complexity, PERC *RECOMMENDS* that endpoints create random SRTP master keys locally to be used for E2E encryption.

This locally-generated SRTP master key is used along with the master salt transmitted to the endpoint from the key distributor via the EKTKey message to encrypt media end-to-end.

Since the media distributor is not involved in E2E functions, it will not create this key nor have access to any endpoint's E2E key. Note, too, that even the key distributor is unaware of the locally-generated E2E keys used by each endpoint.

The endpoint will transmit its E2E key to other endpoints in the conference by periodically including it in SRTP packets in a Full EKT Field. When placed in the Full EKT Field, it is encrypted using the EKT Key provided by the key distributor. The master salt is not transmitted, though, since all endpoints will have received the same master salt via the EKTKey message. The recommended frequency with which an endpoint transmits its SRTP master key is specified in [I-D.ietf-perc-srtp-ekt-diet].

A.4. Who has What Key

All endpoints have knowledge of the KEK.

Every HBH key is distinct for a given endpoint, thus Endpoint A and endpoint B do not have knowledge of the other's HBH key.

Each endpoint generates its own E2E Key (SRTP master key), thus the key distinct per endpoint. This key is transmitted (encrypted) via the EKT Field to other endpoints. Endpoints that receive media from a given transmitting endpoint will therefore have knowledge of the transmitter's E2E key.

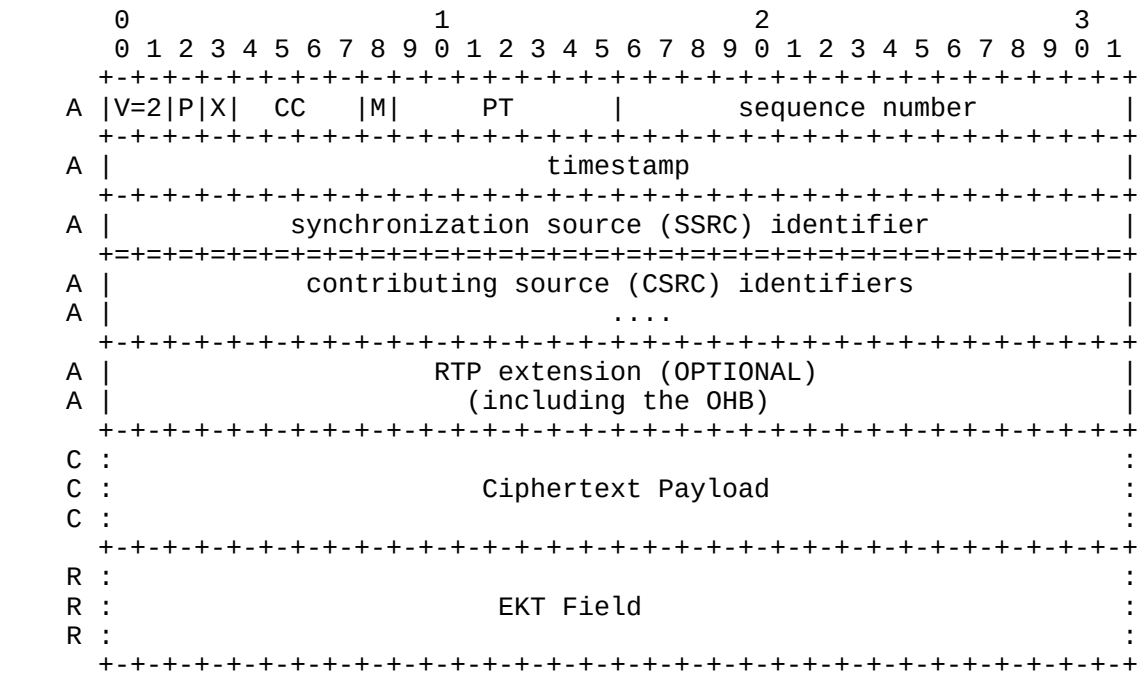
To summarize the various keys and which entity is in possession of a given key, refer to Figure 4.

Key / Entity	Endpoint A	MD X	MD Y	Endpoint B
KEK	Yes	No	No	Yes
E2E Key (A and B)	Yes	No	No	Yes
HBH Key (A<=>MD X)	Yes	Yes	No	No
HBH Key (B<=>MD Y)	No	No	Yes	Yes
HBH Key (MD X<=>MD Y)	No	Yes	Yes	No

Figure 4: Keys per Entity

Appendix B. PERC Packet Format

Figure 5 presents a complete picture of what a PERC packet looks like when transmitted over the wire.



C = Ciphertext (encrypted and authenticated)
A = Associated Data (authenticated only)
R = neither encrypted nor authenticated, added
after Authenticated Encryption completed

Figure 5: PERC Packet Format

Authors' Addresses

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

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

David Benham
Cisco
170 West Tasman Drive
San Jose, California 95134
USA

Email: dbenham@cisco.com

Christian Groves
Independent
Melbourne
Australia

Email: Christian.Groves@nteczone.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

C. Jennings
Cisco Systems
J. Mattsson
Ericsson AB
D. McGrew
D. Wing
F. Andreason
Cisco Systems
March 5, 2018

Encrypted Key Transport for DTLS and Secure RTP
draft-ietf-perc-srtp-ekt-diet-07

Abstract

Encrypted Key Transport (EKT) is an extension to DTLS and 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 for decentralized conferences by distributing a common key to all of the conference 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 https://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 6, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Overview 4
3. Conventions Used In This Document 4
4. Encrypted Key Transport 4
4.1. EKT Field Formats 5
4.2. Packet Processing and State Machine 7
4.2.1. Outbound Processing 8
4.2.2. Inbound Processing 9
4.3. Implementation Notes 10
4.4. Ciphers 10
4.4.1. Ciphers 11
4.4.2. Defining New EKT Ciphers 12
4.5. Synchronizing Operation 12
4.6. Transport 12
4.7. Timing and Reliability Consideration 12
5. Use of EKT with DTLS-SRTP 13
5.1. DTLS-SRTP Recap 14
5.2. SRTP EKT Key Transport Extensions to DTLS-SRTP 14
5.2.1. Negotiating an EKT Cipher 16
5.2.2. Establishing an EKT Key 16
5.3. Offer/Answer Considerations 18
5.4. Sending the DTLS EKTKey Reliably 18
6. Security Considerations 18
7. IANA Considerations 19
7.1. EKT Message Types 19
7.2. EKT Ciphers 20
7.3. TLS Extensions 21
7.4. TLS Handshake Type 21
8. Acknowledgements 21
9. References 21
9.1. Normative References 21
9.2. Informative References 22
Authors' Addresses 23

1. Introduction

Real-time Transport Protocol (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 synchronization source (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 along with the SSRC, rollover counter (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 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 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 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 can be used in conferences where the central media distributor or conference bridge can not decrypt the media, such as the type defined for [I-D.ietf-perc-private-media-framework]. It can also be used for large scale conferences where the conference bridge or media distributor can decrypt all the media but wishes to encrypt the media it is sending just once then send the same encrypted media to a large number of participants. This reduces the amount of CPU time needed for encryption and can be used for some optimization to media sending that use source specific multicast.

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 those methods of the burden to deliver the context for each SRTP source to every SRTP participant.

2. Overview

This specification defines a way for the server in a DTLS-SRTP negotiation to provide an ekt_key to the client during the DTLS handshake. This ekt_key can be used to encrypt the SRTP master key used to encrypt the media the endpoint sends. This specification also defines a way to send the encrypted SRTP master key along with the SRTP packet. Endpoints that receive this and know the ekt_key can use the ekt_key to decrypt the SRTP master key then use the SRTP master key to decrypt the SRTP packet.

One way to use this is used is described in the architecture defined by [I-D.ietf-perc-private-media-framework]. Each participants in the conference call forms a DTLS-SRTP connection to a common key distributor that gives all the endpoints the same ekt_key. Then each endpoint picks there own SRTP master key for the media they send. When sending media, the endpoint also includes the SRTP master key encrypted with the ekt_key. This allows all the endpoints to decrypt the media.

3. 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].

4. Encrypted Key Transport

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

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. Senders MUST NOT include MKI when using EKT. Receivers SHOULD simply ignore any MKI field received if EKT is in use.

4.1. EKT Field Formats

The EKT Field uses the format defined below for the FullEKTField and ShortEKTField.

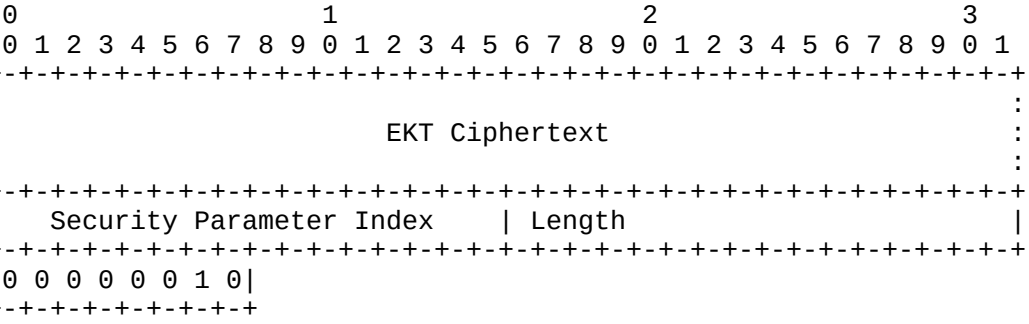


Figure 1: Full EKT Field format

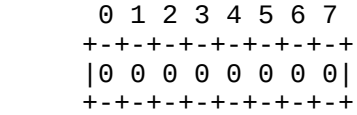


Figure 2: Short EKT Field format

The following shows the syntax of the EKTField expressed in ABNF [RFC5234]. The EKTField is added to the end of an SRTP or SRTCP packet. The EKTPlaintext is the concatenation of SRTPMasterKeyLength, SRTPMasterKey, SSRC, and ROC in that order. The EKTCiphertext is computed by encrypting the EKTPlaintext using the EKTKey. Future extensions to the EKTField MUST conform to the syntax of ExtensionEKTField.

```
BYTE = %x00-FF

EKTMsgTypeFull = %x02
EKTMsgTypeShort = %x00
EKTMsgTypeExtension = %x03-FF

EKTMsgLength = 2BYTE;

SRTPMasterKeyLength = BYTE
SRTPMasterKey = 1*256BYTE
SSRC = 4BYTE; SSRC from RTP
ROC = 4BYTE ; ROC from SRTP FOR THE GIVEN SSRC

EKTPlaintext = SRTPMasterKeyLength SRTPMasterKey SSRC ROC

EKTCiphertext = 1*256BYTE ; EKTEncrypt(EKTKey, EKTPlaintext)
SPI = 2BYTE

FullEKTField = EKTCiphertext SPI EKTMsgLength EKTMsgTypeFull

ShortEKTField = EKTMsgTypeShort

ExtensionData = 1*1024BYTE
ExtensionEKTField = ExtensionData EKTMsgLength EKTMsgTypeExtension

EKTField = FullEKTField / ShortEKTField / ExtensionEKTField
```

Figure 3: EKTField Syntax

These fields and data elements are defined as follows:

EKTPlaintext: 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.

EKTCiphertext: The data that is output from the EKT encryption operation, described in Section 4.4. This field is included in SRTP packets when EKT is in use. The length of EKTCiphertext can be larger than the length of the EKTPlaintext that was encrypted.

SRTPMasterKey: On the sender side, the SRTP Master Key associated with the indicated SSRC.

SRTPMasterKeyLength: The length of the SRTPMasterKey in bytes. This depends on the cipher suite negotiated for SRTP using SDP Offer/ Answer [RFC3264] for the SRTP.

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. The length of this field is 16 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. The Master Salt is communicated separately, via signaling, typically along with the EKTKey.

Together, these data elements are called an EKT parameter set. Each distinct EKT parameter set that is used MUST be associated with a distinct SPI value to avoid ambiguity.

EKTMsgLength: All EKT message other that ShortEKTField have a length as second from the last element. This is the length in octets of either the FullEKTField/ExtensionEKTField including this length field and the following message type.

Message Type: The last byte is used to indicate the type of the EKTField. This MUST be 2 in the FullEKTField format and 0 in ShortEKTField format. Values less than 64 are mandatory to understand while other values are optional to understand. A receiver SHOULD discard the whole EKTField if it contains any message type value that is less than 64 and that is not understood. Message type values that are 64 or greater but not implemented or understood can simply be ignored.

4.2. Packet Processing and State Machine

At any given time, each SRTP/SRTCP source 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.

Either the FullEKTField or ShortEKTField is appended at the tail end of all SRTP packets. The decision on which to send is specified in Section 4.7.

4.2.1. Outbound Processing

See Section 4.7 which describes when to send an SRTP packet with a FullEKTField. If a FullEKTField is not being sent, then a ShortEKTField is sent so the receiver can correctly determine how to process the packet.

When an SRTP packet is sent with a FullEKTField, the EKTField for that packet is created as follows, or uses an equivalent set of steps. The creation of the EKTField MUST precede the normal SRTP packet processing.

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

2. The EKTPlaintext field is computed from the SRTP Master Key, SSRC, and ROC fields, as shown in Section 4.1. 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 EKTCiphertext field is set to the ciphertext created by encrypting the EKTPlaintext with the EKT cipher, using the EKTKey as the encryption key. The encryption process is detailed in Section 4.4.

4. Then the FullEKTField is formed using the EKTCiphertext and the SPI associated with the EKTKey used above. Also appended are the Length and Message Type using the FullEKTField format.

   * Note: the value of the EKTCiphertext field is identical in successive packets protected by the same EKTKey and SRTP master key. This value MAY be cached by an SRTP sender to minimize computational effort.

   The computed value of the FullEKTField is written into the packet.

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

4.2.2. Inbound Processing

When receiving a packet on a RTP stream, the following steps are applied for each received packet.

1. The final byte is checked to determine which EKT format is in use. When an SRTP or SRTCP packet contains a ShortEKTField, the ShortEKTField is removed from the packet then normal SRTP or SRTCP processing occurs. If the packet contains a FullEKTField, then processing continues as described below. The reason for using the last byte of the packet to indicate the type is that the length of the SRTP or SRTCP part is not known until the decryption has occurred. At this point in the processing, there is no easy way to know where the EKT field would start. However, the whole UDP packet has been received so instead of the starting at the front of the packet, the parsing works backwards off the end of the packet and thus the type is put at the very end of the packet.

2. The Security Parameter Index (SPI) field is used to find which EKT parameter set to 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. The EKT parameter set contains the EKTKey, EKTCipher, and SRTP Master Salt.

3. The EKTCiphertext authentication is checked and it is decrypted, as described in Section 4.4, using the EKTKey and EKTCipher found in the previous step. If the EKT decryption operation returns an authentication failure, then the packet processing stops.

4. The resulting EKTPlaintext is parsed as described in Section 4.1, to recover the SRTP Master Key, SSRC, and ROC fields. The SRTP Master Salt that is associated with the EKTKey is also retrieved. If the value of the srtp_master_salt sent as part of the EKTkey is longer than needed by SRTP, then it is truncated by taking the first N bytes from the srtp_master_salt field.

5. If the SSRC in the EKTPlaintext does not match the SSRC of the SRTP packet, then all the information from this EKTPlaintext MUST be discarded and the following steps in this list are not done.

6. The SRTP Master Key, ROC, and SRTP Master Salt from the previous step are saved in a map indexed by the SSRC found in the EKTPlaintext and can be used for any future crypto operations on

the inbound packets with that SSRC. If the SRTP Master Key recovered from the EKTPlaintext is longer than needed by SRTP transform in use, the first bytes are used. If the SRTP Master Key recovered from the EKTPlaintext is shorter than needed by SRTP transform in use, then the bytes received replace the first bytes in the existing key but the other bytes after that remain the same as the old key. This allows for replacing just half the key for transforms such as [I-D.ietf-perc-double]. Outbound packets SHOULD continue to use the old SRTP Master Key for 250 ms after sending any new key. This gives all the receivers in the system time to get the new key before they start receiving media encrypted with the new key.

7. At this point, EKT processing has successfully completed, and the normal SRTP or SRTCP processing takes place including replay protection.

4.3. Implementation Notes

The value of the EKTCiphertext field is identical in successive packets protected by the same EKT parameter set and the same SRTP master key, and ROC. This ciphertext value MAY be cached by an SRTP receiver to minimize computational effort by noting when the SRTP master key is unchanged and avoiding repeating the above steps.

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.

When receiving a new EKTKey, implementations need to use the ekt_ttl to create a time after which this key cannot be used and they also need to create a counter that keeps track of how many times the keys has been used to encrypt data to ensure it does not exceed the T value for that cipher. If either of these limits are exceeded, the key can no longer be used for encryption. At this point implementation need to either use the call signaling to renegotiation a new session or need to terminate the existing session. Terminating the session is a reasonable implementation choice because these limits should not be exceeded except under an attack or error condition.

4.4. Ciphers

EKT uses an authenticated cipher to encrypt and authenticate the EKTPlaintext. This specification defines the interface to the cipher, in order to abstract the interface away from the details of that function. This specification also defines the default cipher

that is used in EKT. The default cipher described in Section 4.4.1 MUST be implemented, but another cipher that conforms to this interface MAY be used.

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

- a secret key K with a length of L bytes, and
- 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 may be larger than M. The decryption function D(K, C) takes the following inputs:

- a secret key K with a length of L bytes, and
- 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. The EKTKey MUST NOT be used for encryption more that T times. Note that if the same FullEKTField is retransmitted 3 times, that only counts as 1 encryption.

Security requirements for EKT ciphers are discussed in Section 6.

4.4.1. Ciphers

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 + (M mod 8) + 8 octets. It can be used with key sizes of L = 16, and L = 32 octets, and its use with those key sizes is indicated as AESKW128, or AESKW256, respectively. The key size determines the length of the AES key used by the Key Wrap algorithm. With this cipher, T=2^48.

Cipher	L	T
AESKW128	16	2^48
AESKW256	32	2^48

Table 1: EKT Ciphers

As AES-128 is the mandatory to implement transform in SRTP, AESKW128 MUST be implemented for EKT and AESKW256 MAY be implemented.

4.4.2. Defining New EKT Ciphers

Other specifications may extend this document by defining other EKTCiphers as described in Section 7. This section defines how those ciphers interact with this specification.

An EKTCipher determines how the EKTCiphertext 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, and T MUST be defined by each EKTCipher. The cipher MUST provide integrity protection.

4.5. Synchronizing Operation

If a source has its EKTKey changed by the key management, it MUST also change its SRTP master key, which will cause it to send out a new FullEKTField. This ensures that if key management thought the EKTKey needs changing (due to a participant leaving or joining) and communicated that 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 EKTKey.

4.6. 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.

4.7. Timing and Reliability Consideration

A system using EKT learns the SRTP master keys distributed with FullEKTFields sent with the SRTP, rather than with call signaling. A

receiver can immediately decrypt an SRTP packet, 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.

The three cases are:

New sender:
: A new sender SHOULD send a packet containing the FullEKTField 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. To accommodate packet loss, it is RECOMMENDED that three consecutive packets contain the FullEKTField be transmitted.

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 FullEKTField. 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 FullEKT Field. If sending audio and video, the RECOMMENDED frequency is the same as the rate of intra coded video frames. If only sending audio, the RECOMMENDED frequency is every 100ms.

5. Use of EKT with DTLS-SRTP

This document defines an extension to DTLS-SRTP called SRTP EKT Key Transport which enables secure transport of EKT keying material from the DTLS-SRTP peer in the server role to the client. This allows 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, which has strong authentication of the endpoint and flexibility, along with allowing secure multiparty RTP with loose coordination and efficient communication of per-source keys.

5.1. 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.

5.2. SRTP EKT Key Transport Extensions to DTLS-SRTP

This document defines a new TLS negotiated extension supported_ekt_ciphers and a new TLS handshake message type ekt_key. The extension negotiates the cipher to be used in encrypting and decrypting EKTCiphertext values, and the handshake message carries the corresponding key.

The diagram below shows a message flow of DTLS 1.3 client and server using EKT configured using the DTLS extensions described in this section. (The initial cookie exchange and other normal DTLS messages are omitted.)

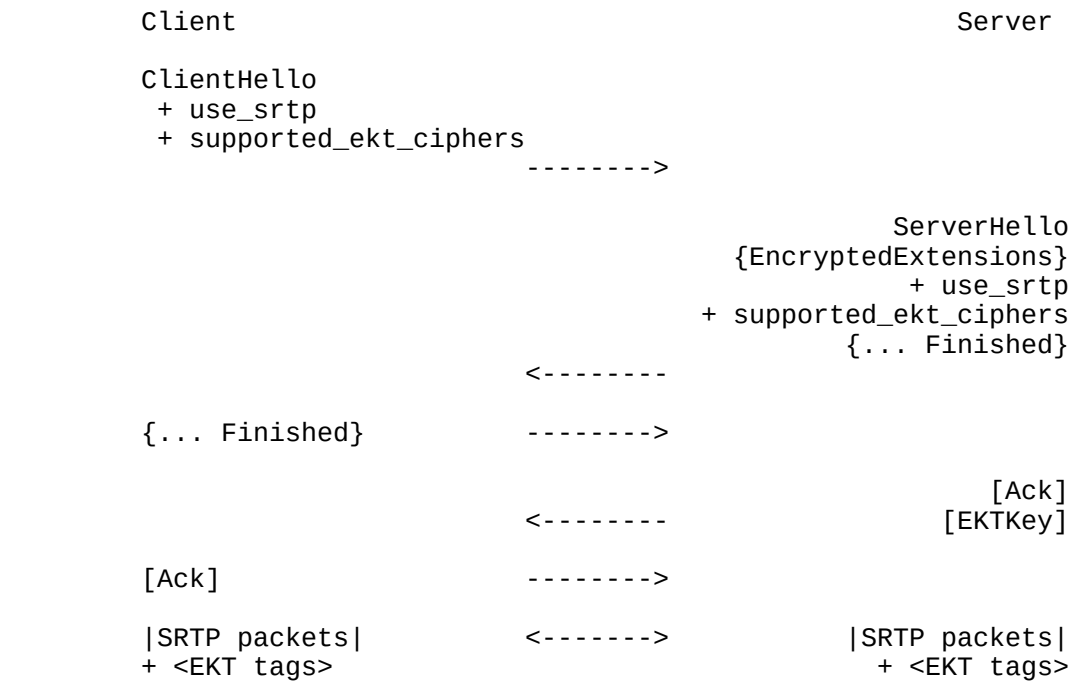
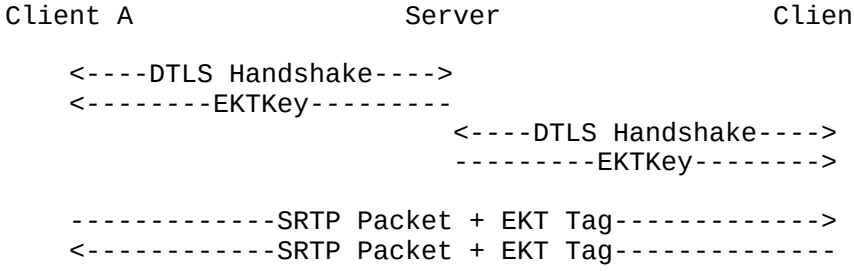


Figure 4

In the context of a multi-party SRTP session in which each endpoint performs a DTLS handshake as a client with a central DTLS server, the extensions defined in this session allow the DTLS server to set a common EKT key among all participants. Each endpoint can then use EKT tags encrypted with that common key to inform other endpoint of the keys it is using to protect SRTP packet. This avoids the need for many individual DTLS handshakes among the endpoints, at the cost of preventing endpoints from directly authenticating one another.



5.2.1. Negotiating an EKT Cipher

To indicate its support for EKT, a DTLS-SRTP client includes in its ClientHello an extension of type supported_ekt_ciphers listing the EKT ciphers the client supports in preference order, with the most preferred version first. If the server agrees to use EKT, then it includes a supported_ekt_ciphers extension in its ServerHello containing a cipher selected from among those advertised by the client.

The extension_data field of this extension contains an "EKTCipher" value, encoded using the syntax defined in [RFC5246]:

```
enum {
  reserved(0),
  aeskw_128(1),
  aeskw_256(2),
} EKTCipherType;

struct {
    select (Handshake.msg_type) {
        case client_hello:
            EKTCipherType supported_ciphers<1..255>;

        case server_hello:
            EKTCipherType selected_cipher;
    };
} EKTCipher;
```

5.2.2. Establishing an EKT Key

Once a client and server have concluded a handshake that negotiated an EKT cipher, the server MUST provide to the client a key to be used when encrypting and decrypting EKTCiphertext values. EKT keys are sent in encrypted handshake records, using handshake type ekt_key(TBD). The body of the handshake message contains an EKTKey structure:

[[NOTE: RFC Editor, please replace "TBD" above with the code point assigned by IANA]]

```
struct {
  opaque ekt_key_value<1..256>;
  opaque srtp_master_salt<1..256>;
  uint16 ekt_spi;
  uint24 ekt_ttl;
} EKTKey;
```

The contents of the fields in this message are as follows:

ekt_key_value
: The EKT Key that the recipient should use when generating EKTCiphertext values

srtp_master_salt
: The SRTP Master Salt to be used with any Master Key encrypted with this EKT Key

ekt_spi
: The SPI value to be used to reference this EKT Key and SRTP Master Salt in EKT tags (along with the EKT cipher negotiated in the handshake)

ekt_ttl
: The maximum amount of time, in seconds, that this EKT Key can be used. The ekt_key_value in this message MUST NOT be used for encrypting or decrypting information after the TTL expires.

If the server did not provide a supported_ekt_ciphers extension in its ServerHello, then EKTKey messages MUST NOT be sent by either the client or the server.

When an EKTKey is received and processed successfully, the recipient MUST respond with an Ack handshake message as described in Section 7 of [I-D.ietf-tls-dtls13]. The EKTKey message and Ack must be retransmitted following the rules in Section 4.2.4 of [RFC6347].

Note: To be clear, EKT can be used with versions of DTLS prior to 1.3. The only difference is that in a pre-1.3 TLS stacks will not have built-in support for generating and processing Ack messages.

If an EKTKey message is received that cannot be processed, then the recipient MUST respond with an appropriate DTLS alert.

5.3. 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.

5.4. Sending the DTLS EKTKey Reliably

The DTLS EKTKey message is sent using the retransmissions specified in Section 4.2.4. of DTLS [RFC6347]. Retransmission is finished with an Ack message or an alert is received.

6. 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.

SRTP master keys MUST be randomly generated, and [RFC4086] offers some guidance about random number generation. SRTP master keys MUST NOT be re-used for any other purpose, and SRTP master keys MUST NOT be derived from other SRTP master keys.

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

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

An attacker who tampers with the bits in FullEKTField can prevent the intended receiver of that packet from being able to decrypt it. This is a minor denial of service vulnerability. Similarly the attacker could take an old FullEKTField from the same session and attach it to the packet. The FullEKTField would correctly decode and pass integrity but the key extracted from the FullEKTField , when used to decrypt the SRTP payload, would be wrong and the SRTP integrity check would fail. Note that the FullEKTField only changes the decryption key and does not change the encryption key. None of these are

considered significant attacks as any attacker that can modify the packets in transit and cause the integrity check to fail.

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. In some cases, caching the previous values of the Ciphertext as described in Section 4.3 helps mitigate this issue.

Each EKT cipher specifies a value T that is the maximum number of times a given key can be used. An endpoint MUST NOT encrypt more than T different Full EKT Field using the same EKTKey. In addition, the EKTKey MUST NOT be used beyond the lifetime provided by the TTL described in Figure 4.

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

Part of the EKTPlaintext 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.

In some systems, when a member of a conference leaves the conferences, the conferences is rekeyed so that member no longer has the key. When changing to a new EKTKey, it is possible that the attacker could block the EKTKey message getting to a particular endpoint and that endpoint would keep sending media encrypted using the old key. To mitigate that risk, the lifetime of the EKTKey SHOULD be limited using the ekt_ttl.

7. IANA Considerations

7.1. EKT Message Types

IANA is requested to create a new table for "EKT Messages Types" in the "Real-Time Transport Protocol (RTP) Parameters" registry. The initial values in this registry are:

Message Type	Value	Specification
Short	0	RFCAAAA
Full	2	RFCAAAA
Reserved	63	RFCAAAA
Reserved	255	RFCAAAA

Table 2: EKT Messages Types

Note to RFC Editor: Please replace RFCAAAA with the RFC number for this specification.

New entries to this table can be added via "Specification Required" as defined in [RFC8126]. When requesting a new value, the requestor needs to indicate if it is mandatory to understand or not. If it is mandatory to understand, IANA needs to allocate a value less than 64, if it is not mandatory to understand, a value greater than or equal to 64 needs to be allocated. IANA SHOULD prefer allocation of even values over odd ones until the even code points are consumed to avoid conflicts with pre standard versions of EKT that have been deployed.

All new EKT messages MUST be defined to have a length as second from the last element.

7.2. EKT Ciphers

IANA is requested to create a new table for "EKT Ciphers" in the "Real-Time Transport Protocol (RTP) Parameters" registry. The initial values in this registry are:

Name	Value	Specification
AESKW128	1	RFCAAAA
AESKW256	2	RFCAAAA
Reserved	255	RFCAAAA

Table 3: EKT Cipher Types

Note to RFC Editor: Please replace RFCAAAA with the RFC number for this specification.

New entries to this table can be added via "Specification Required" as defined in [RFC8126]. The expert SHOULD ensure the specification

defines the values for L and T as required in Section 4.4 of RFCAAAA. Allocated values MUST be in the range of 1 to 254.

7.3. TLS Extensions

IANA is requested to add supported_ekt_ciphers as a new extension name to the "ExtensionType Values" table of the "Transport Layer Security (TLS) Extensions" registry with a reference to this specification and allocate a value of TBD to for this.

[[Note to RFC Editor: TBD will be allocated by IANA.]]

Considerations for this type of extension are described in Section 5 of [RFC4366] and requires "IETF Consensus".

7.4. TLS Handshake Type

IANA is requested to add ekt_key as a new entry in the "TLS HandshakeType Registry" table of the "Transport Layer Security (TLS) Parameters" registry with a reference to this specification, a DTLS-OK value of "Y", and allocate a value of TBD to for this content type.

[[Note to RFC Editor: TBD will be allocated by IANA.]]

This registry was defined in Section 12 of [RFC5246] and requires "Standards Action".

8. Acknowledgements

Thank you to Russ Housley provided detailed review and significant help with crafting text for this document. Thanks to David Benham, Yi Cheng, Lakshminath Dondeti, Kai Fischer, Nermeen Ismail, Paul Jones, Eddy Lem, Jonathan Lennox, Michael Peck, Rob Raymond, Sean Turner, Magnus Westerlund, and Felix Wyss for fruitful discussions, comments, and contributions to this document.

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, <https://www.rfc-editor.org/info/rfc2119>.

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

[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, <https://www.rfc-editor.org/info/rfc3711>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6347>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

9.2. Informative References

[I-D.ietf-perc-double] Jennings, C., Jones, P., Barnes, R., and A. Roach, "SRTP Double Encryption Procedures", draft-ietf-perc-double-07 (work in progress), September 2017.

[I-D.ietf-perc-private-media-framework]
Jones, P., Benham, D., and C. Groves, "A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing", draft-ietf-perc-private-media-framework-05 (work in progress), October 2017.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-22 (work in progress), November 2017.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, DOI 10.17487/RFC4366, April 2006, <https://www.rfc-editor.org/info/rfc4366>.

Authors' Addresses

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

David A. McGrew
Cisco Systems

Email: mcgrew@cisco.com

Dan Wing
Cisco Systems

Email: dwing@cisco.com

Flemming Andreason
Cisco Systems

Email: fandreas@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

P. Jones
Cisco Systems
N. Ohlmeier
Mozilla

March 13, 2017

Transporting the SDP attribute ’dtls-id’ in TLS and DTLS
draft-jones-perc-dtls-id-00

Abstract

This draft defines a new extension to carry the "dtls-id" value defined for use in the Session Description Protocol within TLS and DTLS.

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 14, 2017.

Table of Contents

1. Introduction 2
2. Conventions Used In This Document 2
3. Endpoint procedures 3
4. Media distributor procedures 3
5. Key distributor procedures 3
6. The dtls_id TLS extension 4
7. IANA Considerations 4
8. Security Considerations 4
9. Acknowledgments 5
10. Normative References 5
Authors' Addresses 6

1. Introduction

The Privacy-Enhanced RTP Conferencing (PERC) working group specified a DTLS [RFC6347] tunneling mechanism [I-D.ietf-perc-dtls-tunnel] that enables a media distributor to forward DTLS messages between an endpoint and a key distributor. In the process, the media distributor is able to securely receive only the hop-by-hop keying material, while the endpoints are able to securely receive both end-to-end and hob-by-hop keying material.

An open issue with the current design is how the key distributor can determine which one of several conferences an endpoint is attempting to join. The only information that the key distributor receives via the DTLS tunnel is the endpoint's certificate. However, the same certificate might be used to join several conferences in parallel, thus creating a need for additional information.

[I-D.ietf-mmusic-dtls-sdp] defines an attribute in SDP [RFC4566] called the "dlts-id". The "dtls-id" presented by the endpoint's in SDP will be unique for each DTLS association established using the same certificate. By signaling the certificate fingerprint and "dtls-id" in SDP, along with including the same in the DTLS signaling sent to the key distributor, it would be possible for the key distributor to unambiguously determine which conference key the endpoint should receive.

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.

The terms key distributor, media distributor, endpoint, conference, hop-by-hop keying material, and end-to-end keying material used in this document are introduced in [I-D.ietf-perc-private-media-framework].

3. Endpoint procedures

The endpoint MUST include the "dtls_id" DTLS extension in the "ClientHello" message when establishing a DTLS tunnel in a PERC conference. Likewise, the "dtls-id" SDP attribute MUST be included in SDP sent by the endpoint in both the offer and answer [RFC3264] messages as per [I-D.ietf-mmusic-dtls-sdp].

When receiving a "dtls_id" value from the key distributor, the client MUST check to ensure that value matches the "dtls-id" value received in SDP. If the values do not match, the endpoint MUST consider any received keying material to be invalid and terminate the DTLS association.

4. Media distributor procedures

The media distributor is not required to inspect the "dtls_id" extension, as it merely forwards DTLS messages between the endpoint and the key distributor.

5. Key distributor procedures

This draft assumes that when the endpoint inserts the "dtls-id" into SDP, the information will be conveyed in some way to the key distributor. The process through which the "dtls-id" in SDP is conveyed to the key distributor is outside the scope of this document.

The key distributor MUST extract the "dtls_id" value transmitted in the "ClientHello" message and match that against "dtls-id" value the endpoint transmitted via SDP. If the values in SDP and the "ClientHello" do not match, the DTLS association MUST be rejected.

The key distributor MUST correlate the certificate fingerprint and "dtls_id" received from endpoint’s "ClientHello" message with the corresponding values received from the SDP transmitted by the endpoint. It is through this correlation that the key distributor can be sure to deliver the correct conference key to the endpoint.

When sending the "ServerHello" message, the key distributor MUST insert its own "dtls-id" value. This value MUST also be conveyed back to the client via SDP.

6. The dtls_id TLS extension

The "dtls_id" TLS extension may be used either with TLS [RFC5246] or DTLS. It carries only "dtls-id" value defined in [I-D.ietf-mmusic-dtls-sdp] in the field called "dtls_id". The syntax for the "dtls_id" extension is shown below.

```
struct {
    opaque dtls_id<20..255>;
} SdpDtlsIdData;
```

7. IANA Considerations

This document registers an extension in the TLS "ExtensionType Values" registry established in [RFC5246]. The extension is called "dtls_id" and is assigned the code point TBD. The following addition is made to the registry.

Extension	Recommended	TLS 1.3	HelloRetryRequested
dlts_id	Yes	Encrypted	Yes

8. Security Considerations

The "dtls-id" value is a random value that has no personal identifiable information associated with it. Thus, the value does not expose such information. It also has no particular security properties in and of itself, so being in plaintext in the "ClientHello" or "ServerHello" is not viewed as a security concern.

However, the value does have significance to the receiver, thus changes to the "dtls-id" may result in unexpected behavior. For example, if Alice attempts to join a PERC-enabled conference and the "dtls_id" field is modified in route to the key distributor, Alice may either fail to receive the conference key or receive the wrong conference key. However, since Alice will only be provided keys for conferences for which she is authorized to join based on her client certificate, receiving the wrong key will not compromise the security of the conference. However, receipt of the wrong key will deny Alice access to the plaintext of media transmitted by other participants. Additionally, if Alice transmits media using the wrong conference key, the media will be undecipherable by other conference participants.

As prescribed in these procedures, if the "dtls_id" field transmitted from the key distributor to Alice is modified, Alice will tear down

the DTLS association and fail to join the conference. The result is a denial of service for Alice, but not worse than when any other part of the DTLS message is modified.

9. Acknowledgments

The authors would like to thank Martin Thomson for discussing the idea and providing some initial feedback before the draft was written. We also want to express our appreciation to Cullen Jennings for reviewing the text and providing constructive input.

10. Normative References

[I-D.ietf-mmusic-dtls-sdp]
Holmberg, C. and R. Shpount, "Using the SDP Offer/Answer Mechanism for DTLS", draft-ietf-mmusic-dtls-sdp-20 (work in progress), February 2017.

[I-D.ietf-perc-dtls-tunnel]
Jones, P., Ellenbogen, P., and N. Ohlmeier, "DTLS Tunnel between a Media Distributor and Key Distributor to Facilitate Key Exchange", draft-ietf-perc-dtls-tunnel-00 (work in progress), March 2017.

[I-D.ietf-perc-private-media-framework]
Jones, P., Benham, D., and C. Groves, "A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing", draft-ietf-perc-private-media-framework-02 (work in progress), October 2016.

[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>.

[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>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <http://www.rfc-editor.org/info/rfc4566>.

[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>.

[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

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

Nils H. Ohlmeier
Mozilla

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 11, 2017

A. Roach
Mozilla

March 10, 2017

Using Privacy Enhanced Real-time Conferencing (PERC) in a WebRTC Context
draft-roach-perc-webrtc-00

Abstract

The Privacy-Enhanced Realtime Conferencing (PERC) architecture defines a system by which multiparty conferences can be handled by a conference server that is "semi-trusted": that is, it is trusted to correctly handle media packets, but it is not trusted with the actual contents of the media streams. In order to use this architecture within WebRTC, we must describe how information is conveyed among various network functions. This document describes the information to be conveyed and how it is transferred.

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 11, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Goals 3
4. Data Flows 3
4.1. Key Distributor Co-Located with Browser 3
4.1.1. Conference Establishment 5
4.1.2. Owner Sends Offer to Conference 7
4.1.3. Conference Processes Owner Offer 9
4.1.4. Owner Processes Answer 10
4.1.5. Owner sets up media connection 11
4.1.6. Participant Joins Conference 11
4.1.7. Conference Processes Participant Offer 13
4.1.8. Participant Processes Answer 14
4.1.9. Participant sets up media connection 15
4.2. Key Distributor Separate From Browser 16
5. New Mechanisms Required 16
5.1. New Key Distributor DOM Object 16
5.2. New "sign" Tunnel Message 17
6. Security Considerations 17
7. IANA Considerations 17
8. References 17
8.1. Normative References 17
8.2. Informative References 18
Author's Address 18

1. Introduction

The Privacy-Enhanced Realtime Conferencing (PERC) architecture [I-D.ietf-perc-private-media-framework] defines a system by which multiparty conferences can be handled by a conference server that is "semi-trusted": that is, it is trusted to correctly handle media packets, but it is not trusted with the actual contents of the media streams. In order to use this architecture within WebRTC [W3C.WD-webrtc-20170313], we must describe how information is conveyed among various network functions. This document describes the information to be conveyed and how it is transferred.

Note that the current document is a fairly high-level thumbnail sketch of information flow. It will be expanded with further detail once we have consensus on the general direction for the mechanism.

2. Terminology

This document also uses a number of terms defined in section 2 of [I-D.ietf-perc-private-media-framework]. Of particular note, the definitions for "Media Distributor (MD)" and "Key Distributor (KD)" are provided by that document.

3. Goals

A key goal of the mechanisms described in this document is that it should work with a minimal set of changes to web browsers that already implement WebRTC. In particular: PERC requires the use of strong identity assertions in order to provide any value whatsoever. WebRTC already contains an identity mechanism [I-D.ietf-rtcweb-security-arch]; we do not seek to introduce a second one. Regarding identity, this document assumes:

- From a WebRTC perspective, the remote identity presented to each conference participant will be the identity of the Key Distributor (KD). Whether this is identical to the PeerConnection identity presented by the conference owner is a matter of policy for the conference application. (Note that this is an emergent property of the system rather than a design decision, since the DTLS associations used to key the SRTP terminate at the KD in the PERC architecture.)

- The KD will validate the identity of each user as they join the conference. The roster of current users must be available to at least one of the participants through a means that can be trusted as being authentically generated by the KD.

We also seek to support a system that allows for the Key Distributor (KD) to be co-located with an endpoint as a primary goal, and for the KD to be located on a dedicated server as an important secondary goal.

4. Data Flows

4.1. Key Distributor Co-Located with Browser

When the KD is co-located with the browser, the browser that serves the role of KD must join the conference before any other media can be exchanged. It is up to the application to determine whether this is simply the first participant to join, or a specific participant who has an administrative relationship with the conference instance. For concision, we will use the term "conference owner" to refer to the browser serving in this role, even though its assignment may be arbitrary.

We further posit that browsers that serve as KDs will have specific objects available as part of their Document Object Model (DOM). These objects will give applications the ability to instantiate and control such KDs, but without actually being able to obtain access to the keying material itself. The definition of the interface for such objects is not defined in this document more than necessary. We envision a companion document, submitted to the W3C, to describe this API in detail.

The following callflows illustrate the information exchanged among these entities:

Identity Service: This is the server that acts as in Identity Provider (IdP), as that term is defined in [I-D.ietf-rtcweb-security-arch]. Note that the Owner and the Participant may use different Identity Services to vouch for their identities.

Owner Browser: This is the web browser execution environment for the web browser that is acting as the KD.

Owner PeerConnection: This is the RTCPeerConnection object created and used by conferencing web application running in the Owner Browser. It is shown separately from the Owner Browser to clearly indicate those tasks which are performed directly by the RTCPeerConnection without the ability for the conferencing web application to intervene.

KD: This is the Key Distributor function created and used by the conferencing web application running in the Owner Browser. It is shown separately from the Owner Browser to clearly indicate those tasks which are performed directly by the Key Distributor without the ability for the conferencing web application to intervene.

HTTP Server: This is the HTTP server that handles the signaling for the conference.

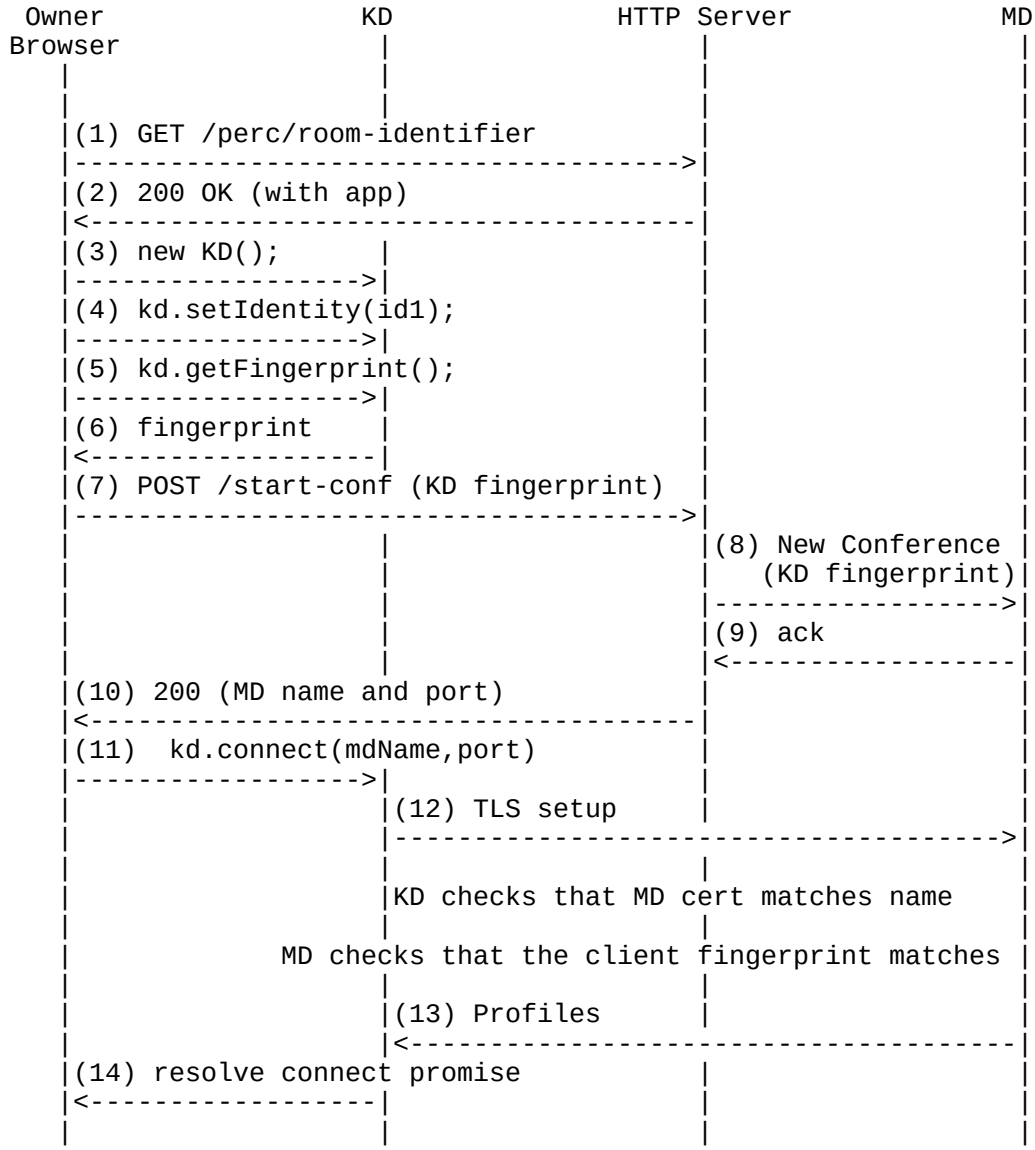
MD: This is the media distributor that handles the media the conference. Note that the interface between the HTTP Server and the MD is an implementation detail for the developer of the conference server. The messages exchanged between them are intended to illustrate the information required to be exchanged between them to have a properly functioning PERC conference.

Participant Browser: This is the web browser execution environment for a web browser that is not acting as the KD.

Participant PeerConnection: This is the RTCPeerConnection object created and used by conferencing web application running in the Participant Browser. It is shown separately from the Participant Browser to clearly indicate those tasks which are performed directly by the RTCPeerConnection without the ability for the conferencing web application to intervene.

The following sub sections step through an illustrative call flow that demonstrates how the information needed to create a PERC conference in a WebRTC environment is exchanged among the aforementioned functions. The call flows occur in the order shown by these sections - division into sections is done primarily to limit the number of elements in any diagram to no more than four, since the limitations of the internet-draft format makes wider diagrams impossible.

4.1.1. Conference Establishment



1. The conference owner loads the conference application. Although not required, information sufficient to identify the conference is frequently sent as part of the URL.

2. The HTTP server returns the conferencing web application.

3. The conferencing web application instantiates a new Key Distributor (KD) object. Upon instantiation, this KD creates a new certificate.

4. The conferencing web app sets the owner’s identity on the KD.

5. The application asks for the certificate’s fingerprint...

6. ...which the KD provides

7. The application initiates the conference, including the KD cert fingerprint as part of the initiation message.

8. The HTTP server passes the KD fingerprint along to the Media Distributor (MD). This is used later by the MD to ensure that the correct KD is connecting to it.

9. The MD acknowledges creation of a new conference. This acknowledgement contains the hostname and port that the MD is listening on for the KD to connect to.

10. The web server responds to the conferencing web app with the hostname of the MD as well as the port it is listening on for the KD to connect.

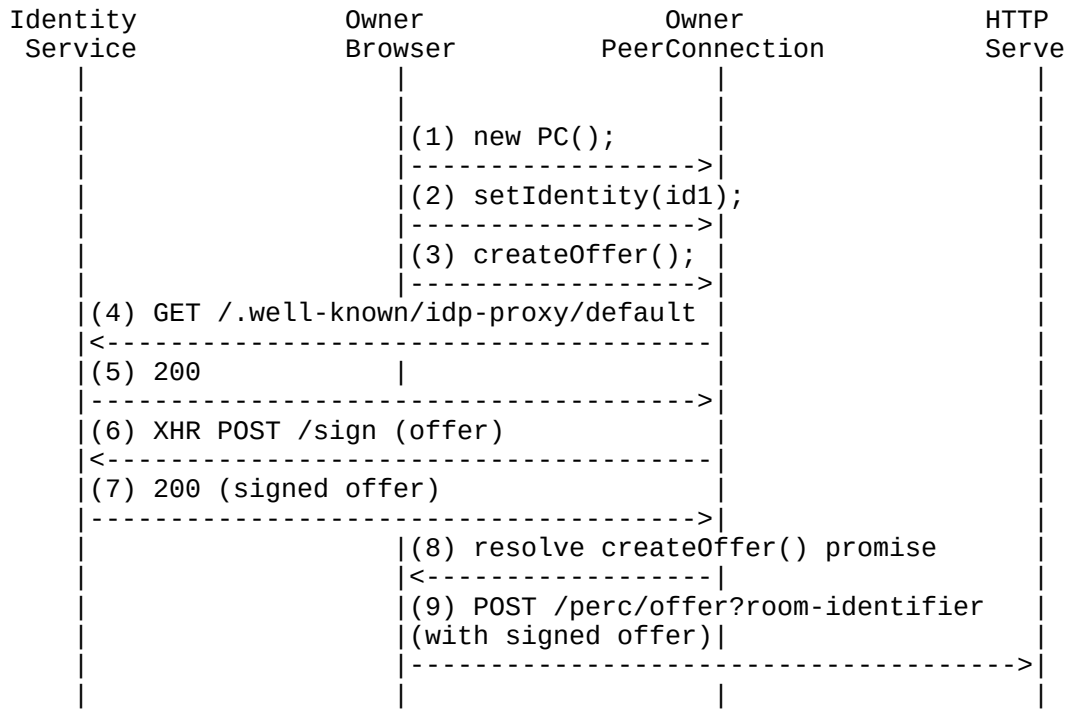
11. The application passes along the MD name and port to the KD object.

12. The KD initiates a TLS connection to the MD. This connection uses the protocol defined in [I-D.ietf-perc-dtls-tunnel]. The KD verifies that the certificate presented by the MD matches the name it used to connect to it, and that it chains up to a trusted certificate authority. The MD verifies that the client cert provided by the KD matches the fingerprint that it received earlier from the HTTP server.

13. The MD returns its list of supported profiles, as described in [I-D.ietf-perc-dtls-tunnel].

14. The KD object indicates to the conferencing app that the KD-MD connection has been successfully established.

4.1.2. Owner Sends Offer to Conference



1. The conferencing web application creates a new WebRTC RTCPeerConnection (PC) object to allow sending and receiving media.

2. The conferencing web app sets the owner’s identity on the PC...

3. ...and requests an offer to be created.

4. As described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy code from the Identity Service...

5. ...which it returns.

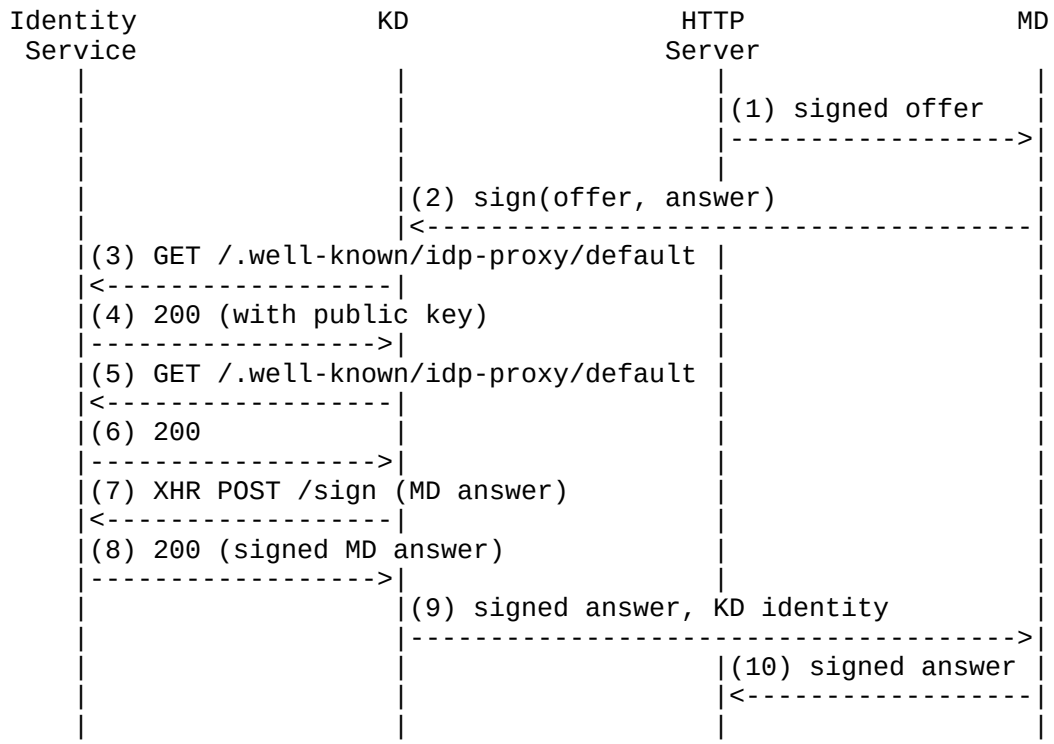
6. Upon being executed, the idp-proxy code sends the offer to the Identity service...

7. ...which verifies user’s identity, the signs the offer, and returns the signed offer to the PC.

8. The PC then returns the signed offer to the conferencing web app.

9. The conferencing web app sends the signed offer to the HTTP server to begin establishing the media connection.

4.1.3. Conference Processes Owner Offer



In this section of the flow, the MD sends generates an answer. It sends this answer to the KD so that it can sign it with an assertion of the KD's identity. It also sends the corresponding offer to the KD so that the KD can validate that the authenticated party who generated the offer is authorized to join the conference, and to allow the KD to add that user to its own notion of the current conference roster.

1. The HTTP server sends the signed offer to the MD to allow it to start setting up the media connection.

2. The MD creates an answer to the received offer, and sends both offer and answer over the MD-KD tunnel connection.

3. Similar to the PC validation procedure described in [I-D.ietf-rtcweb-security-arch], the KD requests the IDP proxy code from the Identity Service based on the identity in the offer...

4. ...which it returns. The KD uses the IDP proxy code to validate the identity of the party that generated the offer.

5. Similar to the PC signing procedure described in [I-D.ietf-rtcweb-security-arch], the KD requests the IDP proxy code from the Identity Service...

6. ...which it returns.

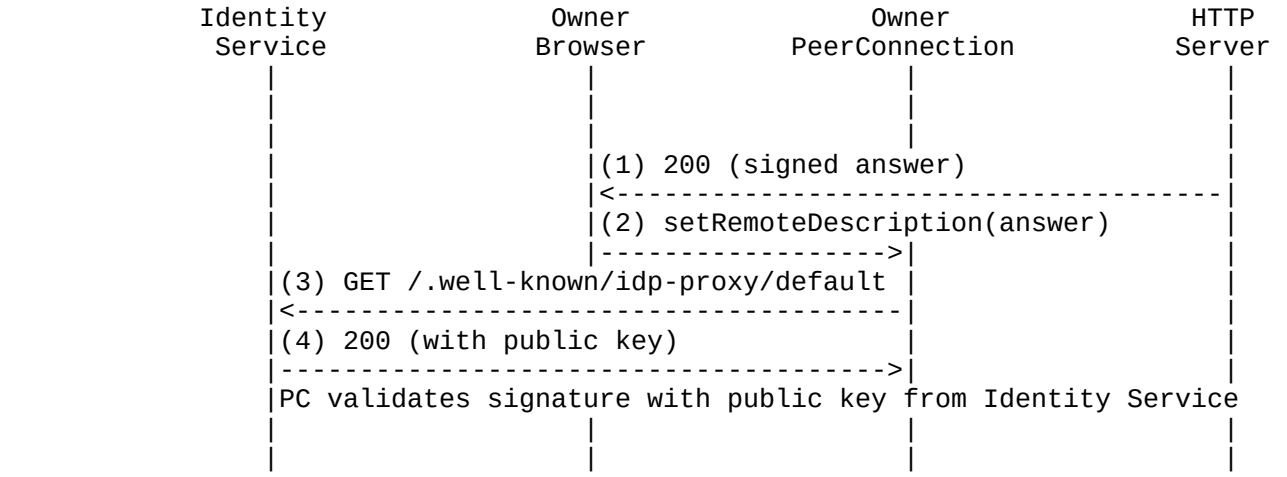
7. Upon being executed, the idp-proxy code sends the MD answer to the Identity Service...

8. ...which verifies KD's identity, the signs the MD answer, and returns the signed MD answer to the KD.

9. The KD sends the signed MD answer back to the MD.

10. The MD sends the signed answer to the HTTP server.

4.1.4. Owner Processes Answer



1. To complete the offer/answer exchange, the HTTP server returns the signed answer (asserting the KD's identity) to the owner's conference web app.

2. The conference web app sets the remote description on the PC to the received answer

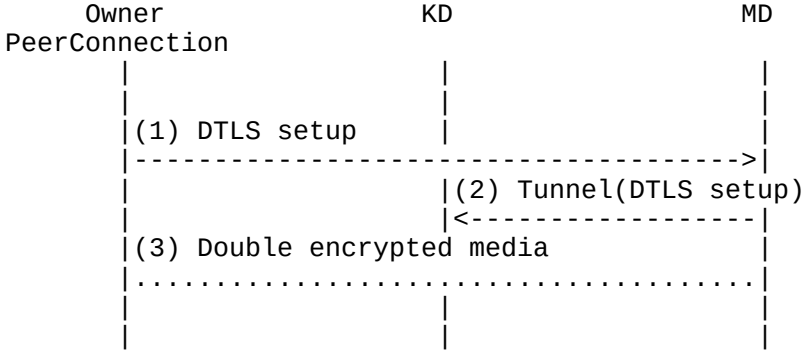
3. Using the identity validation procedure described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy

code from the Identity Service based on the identity in the answer...

4. ...which it returns. The PC uses the IDP proxy code to validate the identity of the party that generated the answer.

4.1.5. Owner sets up media connection

Note: ICE/ICE Lite is not shown in this flow, but is assumed to have taken place.



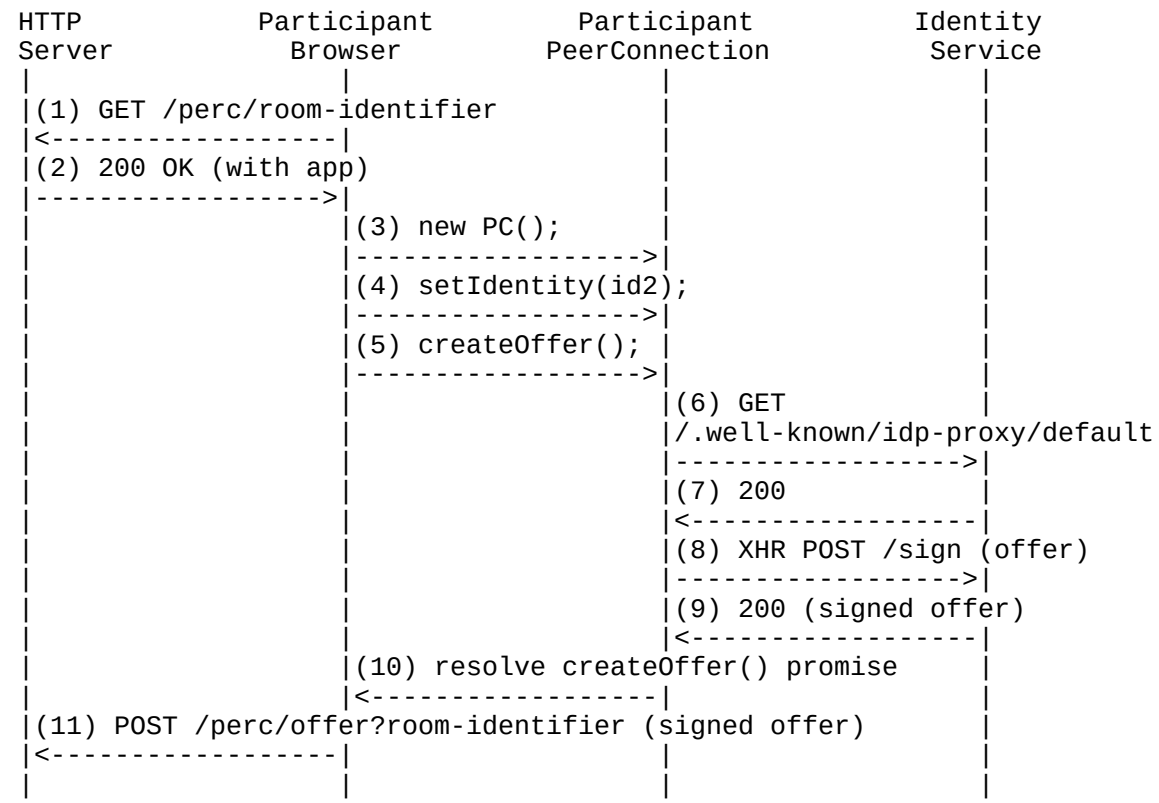
1. The PC, using the addressing information present in the answer, negotiates a DTLS association towards the MD. We make an assumption that the SDP generated by the MD contains a port number that is unique to the conference, allowing it to correlate the incoming DTLS messages to the correct KD.

2. The MD uses the tunneling protocol defined in [I-D.ietf-perc-dtls-tunnel] to forward the DTLS setup messages between the PC and the KD. These DTLS setup messages make use of the mechanism described in [I-D.ietf-perc-srtp-ekt-diet] to establish end-to-end keys for the media.

3. Using the mechanism described in [I-D.ietf-perc-double], the PC now begins to send and received SRTP-encrypted media to and from the MD.

4.1.6. Participant Joins Conference

After the conference has been established as described in the preceding sections, each new participant triggers a flow that closely mirrors the owner PC’s interaction with the conference. These steps are shown in the following sections.



1. The conference owner loads the conference application. Although not required, information sufficient to identify the conference is frequently sent as part of the URL.

2. The HTTP server returns the conferencing web application.

3. The conferencing web application creates a new WebRTC RTCPeerConnection (PC) object to allow sending and receiving media.

4. The conferencing web app sets the owner’s identity on the PC...

5. ...and requests an offer to be created.

6. As described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy code from the Identity Service...

7. ...which it returns.

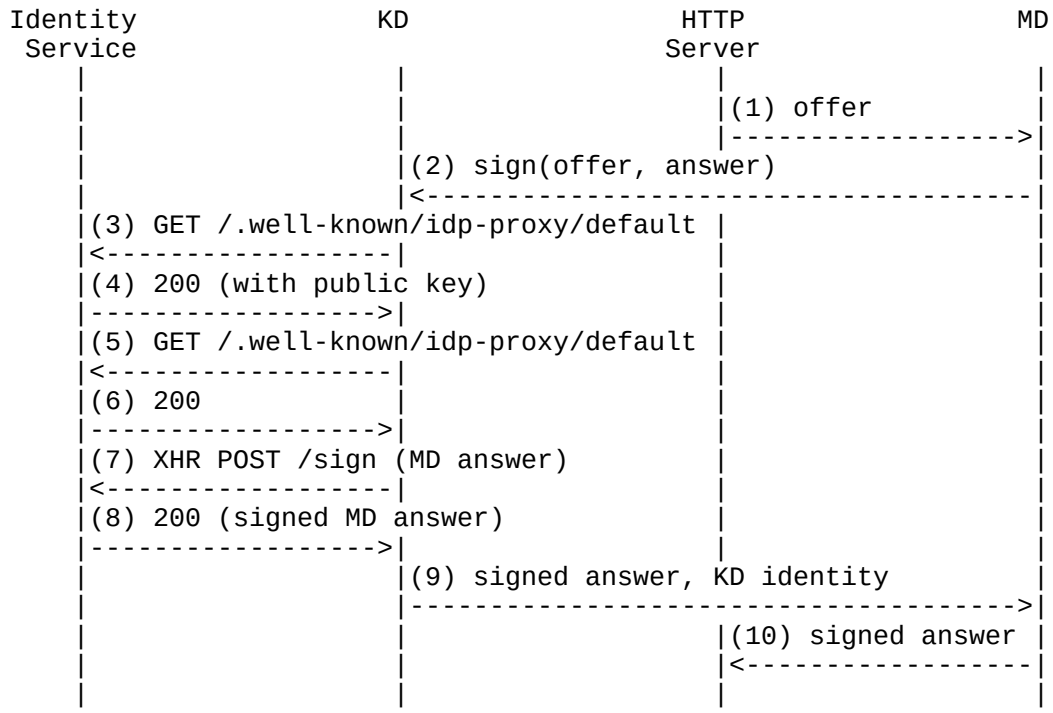
8. Upon being executed, the idp-proxy code sends the offer to the Identity service...

9. ...which verifies user's identity, the signs the offer, and returns the signed offer to the PC.

10. The PC then returns the signed offer to the conferencing web app.

11. The conferencing web app sends the signed offer to the HTTP server to begin establishing the media connection.

4.1.7. Conference Processes Participant Offer



This flow is identical to conference processing of the owner's offer. It is included here for completeness.

1. The HTTP server sends the signed offer to the MD to allow it to start setting up the media connection.

2. The MD creates an answer to the received offer, and sends both offer and answer over the MD-KD tunnel connection.

3. Similar to the PC validation procedure described in [I-D.ietf-rtcweb-security-arch], the KD requests the IDP proxy code from the Identity Service based on the identity in the offer...

4. ...which it returns. The KD uses the IDP proxy code to validate the identity of the party that generated the offer.

5. Similar to the PC signing procedure described in [I-D.ietf-rtcweb-security-arch], the KD requests the IDP proxy code from the Identity Service...

6. ...which it returns.

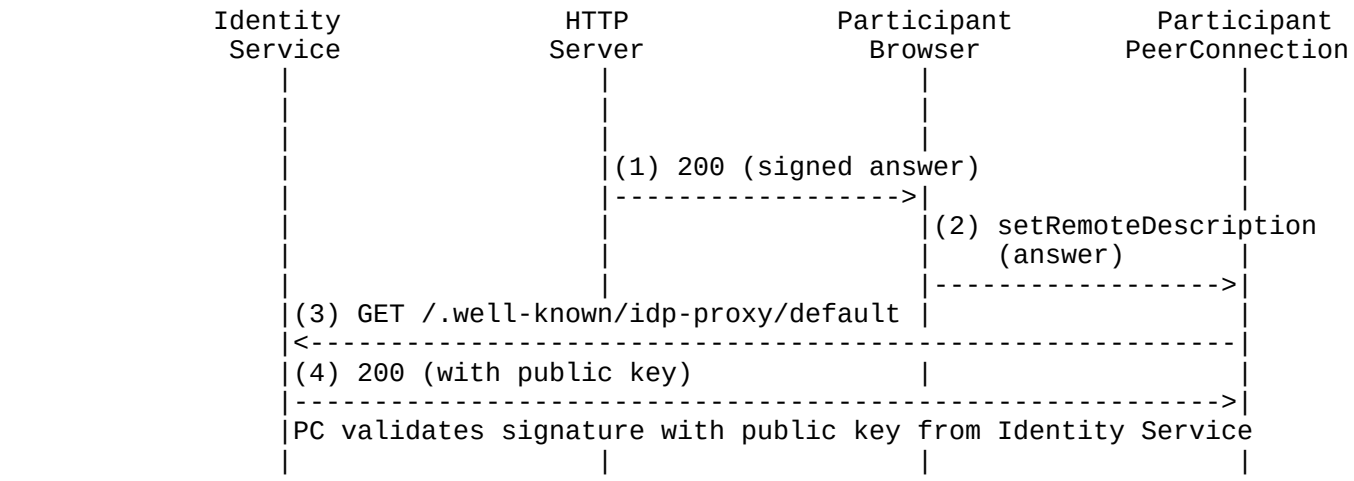
7. Upon being executed, the idp-proxy code sends the MD answer to the Identity Service...

8. ...which verifies KD's identity, the signs the MD answer, and returns the signed MD answer to the KD.

9. The KD sends the signed MD answer back to the MD.

10. The MD sends the signed answer to the HTTP server.

4.1.8. Participant Processes Answer



This flow is identical to owner's processing of the conference's answer. It is included here for completeness.

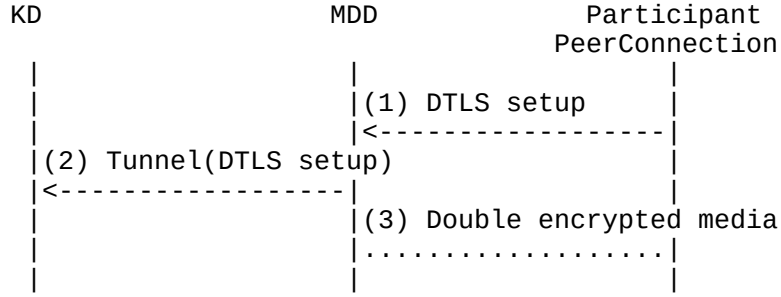
1. To complete the offer/answer exchange, the HTTP server returns the signed answer (asserting the KD's identity) to the owner's conference web app.

2. The conference web app sets the remote description on the PC to the received answer

3. Using the identity validation procedure described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy code from the Identity Service based on the identity in the answer...

4. ...which it returns. The PC uses the IDP proxy code to validate the identity of the party that generated the answer.

4.1.9. Participant sets up media connection



This flow is identical to the owner's establishment of encrypted media. It is included here for completeness.

1. The PC, using the addressing information present in the answer, negotiates a DTLS association towards the MD. We make an assumption that the SDP generated by the MD contains a port number that is unique to the conference, allowing it to correlate the incoming DTLS messages to the correct KD.

2. The MD uses the tunneling protocol defined in [I-D.ietf-perc-dtls-tunnel] to forward the DTLS setup messages between the PC and the KD. These DTLS setup messages make use of the mechanism described in [I-D.ietf-perc-srtp-ekt-diet] to establish end-to-end keys for the media.

3. Using the mechanism described in [I-D.ietf-perc-double], the PC now begins to send and received SRTP-encrypted media to and from the MD.

4.2. Key Distributor Separate From Browser

This configuration introduces a couple of challenges. It is not explored in depth in this version of the document; however, we highlight the following two issues:

- Because identity in WebRTC is inherently based on evaluation of Javascript, and because the KD must validate identities to perform its intended purpose, the KD will be required to include a Javascript execution environment.

- Because the KD is the only trusted node that inherently has access to the list of active conference participants, we must introduce additional mechanisms that allow it to convey this information to conference participants in a way that can be authenticated.

5. New Mechanisms Required

Based on the foregoing message flows, we need to add a small number of new mechanisms to the overall system.

5.1. New Key Distributor DOM Object

Although the formal definition of the Key Distributor DOM object will be provided by a W3C document, the foregoing message flows imply that we will need an interface that looks roughly like the following. This interface is expressed using the syntax defined by [W3C.CR-WebIDL-1-20160308], and refers to a number of structures defined in [W3C.WD-webrtc-20170313].

```
interface RTCKeyDistributor : EventTarget {
    void setIdentityProvider(DOMString provider,
                         optional RTCIdentityProviderOptions options);

    Promise<DOMString> getIdentityAssertion();
    readonly attribute Promise<RTCIdentityAssertion> peerIdentity;
    readonly attribute DOMString?                    idpLoginUrl;
    readonly attribute DOMString?                    idpErrorInfo;

    Promise<RTCCertificate> getCertificate();

    Promise<void> connect(DOMString mdHost, unsigned short mdPort);

    attribute EventHandler              onfingerprintfailure;
}
```

5.2. New "sign" Tunnel Message

We also need to add new "sign_answer" and "sign_answer_ack" MsgTypes to the protocol defined in [I-D.ietf-perc-dtls-tunnel]. These are used by the MD to request that the KD sign its answer.

The "SignAnswer" message is defined as follows:

```
struct {
  opaque offer<1..2^24-1>;
  opaque answer<1..2^24-1>;
} SignAnswer;
```

The "SignAnswerAck" message is defined as follows:

```
struct {
  opaque answer<1..2^24-1>;
} SignAnswerAck;
```

6. Security Considerations

This section will be fleshed out further after the working group has general agreement about the direction this mechanism should take.

The means by which the KD determines that the offer being presented in the "sign" message corresponds to the current conference and has not been replayed has not yet been analyzed.

We need to ensure that MD answers signed by the KD cannot be used by the MD as offers in other contexts, as doing so would allow the MD to impersonate the KD’s identity.

7. IANA Considerations

This document requires no actions from IANA.

8. References

8.1. Normative References

[I-D.ietf-perc-double]
Jennings, C., Jones, P., and A. Roach, "SRTP Double Encryption Procedures", draft-ietf-perc-double-02 (work in progress), October 2016.

[I-D.ietf-perc-dtls-tunnel]
Jones, P., Ellenbogen, P., and N. Ohlmeier, "DTLS Tunnel between a Media Distributor and Key Distributor to Facilitate Key Exchange", draft-ietf-perc-dtls-tunnel-00 (work in progress), March 2017.

[I-D.ietf-perc-srtp-ekt-diet]
Jennings, C., Mattsson, J., McGrew, D., and D. Wing, "Encrypted Key Transport for Secure RTP", draft-ietf-perc-srtp-ekt-diet-02 (work in progress), October 2016.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-12 (work in progress), June 2016.

[W3C.CR-WebIDL-1-20160308]
McCormack, C. and B. Zbarsky, "WebIDL Level 1", World Wide Web Consortium CR CR-WebIDL-1-20160308, March 2016, <http://www.w3.org/TR/2016/CR-WebIDL-1-20160308>.

[W3C.WD-webrtc-20170313]
Bergkvist, A., Burnett, D., Jennings, C., Narayanan, A., and B. Aboba, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20170313, March 2017, <https://www.w3.org/TR/2017/WD-webrtc-20170313>.

8.2. Informative References

[I-D.ietf-perc-private-media-framework]
Jones, P., Benham, D., and C. Groves, "A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing", draft-ietf-perc-private-media-framework-02 (work in progress), October 2016.

Author’s Address

Adam Roach
Mozilla
\
Dallas
US

Phone: +1 650 903 0800 x863
Email: adam@nostrum.com