Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 1, 2020

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

August 29, 2019

SRTP Double Encryption Procedures
draft-ietf-perc-double-12

Abstract

In some conferencing scenarios, it is desirable for an intermediary to be able to manipulate some parameters in Real Time Protocol (RTP) packets, while still providing strong end-to-end security guarantees. This document defines a cryptographic transform for the Secure Real Time Protocol (SRTP) that uses 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 (AEAD) algorithm or take advantage of future SRTP transforms with different properties.

Status of This Memo

Copyright Notice

Table of Contents

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 that enables encryption and authentication of the media end-to-end while still allowing certain information in the header of a Real Time Protocol (RTP) packet 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 a transform for the Secure Real Time Protocol (SRTP) 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. In that case, the original value of any RTP header field that is changed is included in an "Original Header Block" that is added to the packet. 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, including the media payload.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Terms used throughout this document include:

- Media Distributor: A device that receives media from endpoints and distributes it to other endpoints, but does not need to interpret or change the media content (see also [I-D.ietf-perc-private-media-framework])

- end-to-end: The path from one endpoint through one or more Media Distributors to the endpoint at the other end.

- hop-by-hop: The path from the endpoint to or from the Media Distributor.

- o Original Header Block (OHB): 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:

- o 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

- o 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:

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

- o 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.

- o 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.

- o the SSRC is the same for both the inner and out outer algorithms as it can not be changed.

- o The SEQ and ROC are tracked independently for the inner and outer algorithms.

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

Although SRTP uses a single master key to derive keys for an SRTP session, this transform requires separate inner and outer keys. 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 pseudo-random function (PRF), which preserves the separation between the two halves of the key. Given a positive integer "n" representing the desired output length, a master key "k_master", and an input "x":

```
PRF\_double\_n(k\_master,x) = PRF\_(n/2)(inner(k\_master),x) ||
                              PRF\_(n/2)(outer(k\_master),x)
```

Here "PRF_n(k, x)" represents the AES_CM PRF KDF (see Section 4.3.3 of [RFC3711]) for DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM algorithm and AES_256_CM_PRF KDF [RFC6188] for DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM algorithm. "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 RTP header fields. In the encryption process, the OHB is included in an SRTP packet as described in Section 5. 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 [RFC5234]):

```
OCTET = %x00-FF

PT = OCTET
SEQ = 2OCTET
Config = OCTET
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.

If the marker bit is not present (M=0), then B MUST be set to zero. That is, if "C" represents the value of the config octet, then the masked value "C & 0x0C" MUST NOT have the value "0x80".

5. RTP Operations

As implied by the use of the word "double" above, this transform applies AES-GCM to the SRTP packet twice. This allows media distributors to be able to modify some header fields while allowing endpoints to verify the end-to-end integrity of a packet.

The first, "inner" application of AES-GCM encrypts the SRTP payload and integrity-protects a version of the SRTP header with extensions truncated. Omitting extensions from the inner integrity check means that they can be modified by a media distributor holding only the "outer" key.

The second, "outer" application of AES-GCM encrypts the ciphertext produced by the inner encryption (i.e., the encrypted payload and authentication tag), plus an OHB that expresses any changes made between the inner and outer transforms.

A media distributor that has the outer key but not the inner key may modify the header fields that can be included in the OHB by decrypting, modifying, and re-encrypting the packet.

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 (i.e., keep only the first 12 + 4 * CC bytes of the header)

   * Payload: The RTP payload of the original packet (including padding when present)

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 (to restore any header extensions and reset the X bit), and append an empty OHB (0x00) to the encrypted payload (with the authentication tag) obtained from the step 4.

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) cryptographic algorithm, 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 received packet, modifies the packet, updates the OHB with any modifications not already present in the OHB, and re-encrypts the packet using the the outer (hop-by-hop) cryptographic key before transmitting.

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. Make any desired changes to the fields are allowed to be changed, i.e., PT, SEQ, and M. The Media Distributor MAY also make modifications to header extensions, without the need to reflect these changes in the OHB.

3. Reflect any changes to header fields in the OHB:

   * If Media Distributor changed a field that is not already in the OHB, then it MUST add the original value of the field to the OHB. Note that this might result in an increase in the size of the OHB.

   * If the Media Distributor took a field that had previously been modified and reset to its original value, then it SHOULD drop the corresponding information from the OHB. Note that this might result in a decrease in the size of the OHB.

   * Otherwise, the Media Distributor MUST NOT modify the OHB.

4. 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.

In order to avoid nonce reuse, the cryptographic contexts used in step 1 and step 5 MUST use different, independent master keys. Note that this means that the key used for decryption by the MD MUST be different from the key used for re-encryption to the end recipient.

Note that if multiple MDs modify the same packet, then the first MD to alter a given header field is the one that adds it to the OHB. If a subsequent MD changes the value of a header field that has already been changed, then the original value will already be in the OHB, so no update to the OHB is required.

A Media Distributor that decrypts, modifies, and re-encrypts packets in this way MUST use an independent key for each recipient, and MUST NOT re-encrypt the packet using the sender's keys. If the Media Distributor decrypts and re-encrypts with the same key and salt, it will result in the reuse of a (key, nonce) pair, undermining the security of AES-GCM.

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 (i.e., keep only the first 12 + 4 * CC bytes of the header)

* 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:

o The PT from the outer SRTP packet is used for normal matching to SDP and codec selection.

o The sequence number from the outer SRTP packet is used for normal RTP ordering.

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

If the RTP header of the outer packet contains extensions, they MAY be used. However, because extensions are not protected end-to-end, implementations SHOULD reject an RTP packet containing headers that would require end-to-end protection.

6. RTCP Operations

Unlike RTP, which is encrypted both hop-by-hop and end-to-end using two separate cryptographic keys, 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

Media Distributors sometimes interact with RTP media packets sent by endpoints, e.g., to provide recovery or receive commands via DTMF. When media packets are encrypted end-to-end, these procedures require modification. (End-to-end interactions, including end-to-end recovery, are not affected by end-to-end encryption.)

Repair mechanisms, in general, will need to perform recovery on encrypted packets (double-encrypted when using this transform), since the Media Distributor does not have access to the plaintext of the packet, only an intermediate, E2E-encrypted form.

When the recovery mechanism calls for the recovery packet itself to be encrypted, it is encrypted with only the outer, hop-by-hop key. This allows a media distributor to generate recovery packets without having access to the inner, end-to-end keys. However, it also results in recovery packets being triple-encrypted, twice for the base transform, and once for the recovery protection.

7.1. RTP Retransmission (RTX)

When using RTX [RFC4588] with double, the cached payloads MUST be the double-encrypted packets, i.e., the bits that are sent over the wire to the other side. When encrypting a retransmission packet, it MUST be encrypted the packet in repair mode (i.e., with only the hop-by-hop key).

If the Media Distributor were to cache the inner, E2E-encrypted payload and retransmit that with an RTX OSN field prepended, then the modifications to the payload would cause the inner integrity check to fail at the receiver.

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

7.2. Redundant Audio Data (RED)

When using RED [RFC2198] with double, the processing at the sender and receiver is the same as when using RED with any other SRTP transform.

The main difference between double and any other transform is that in an intermediated environment, usage of RED must be end-to-end. A Media Distributor cannot synthesize RED packets, because it lacks access to the plaintext media payloads that are combined to form a RED payload.

Note that FlexFEC may often provide similar or better repair capabilities compared to RED. For most applications, FlexFEC is a better choice than RED; in particular, FlexFEC has modes in which the Media Distributor can synthesize recovery packets.

7.3. Forward Error Correction (FEC)

When using Flex FEC [I-D.ietf-payload-flexible-fec-scheme] with double, repair packets MUST be constructed by first double-encrypting the packet, then performing FEC. Processing of repair packets proceeds in the opposite order, performing FEC recovery and then decrypting. This ensures that the original media is not revealed to the Media Distributor but at the same time allows the Media Distributor to repair media. When encrypting a packet that contains the Flex FEC data, which is already encrypted, it MUST be encrypted with only the outer, hop-by-hop transform.

The algorithm recommended 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 using the mechanism in [RFC4733], it is end-to-end encrypted and the relay can not read it, so it cannot 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 algorithms 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 outer transform as the payload data is already encrypted by the inner transform.

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. The OHB will consume an additional 1-4 octets. Packets in repair mode will carry additional repair data, further increasing their size.

9. Security Considerations

This SRTP transform provides protection against two classes of attacker: An network attacker that knows neither the inner nor outer keys, and a malicious MD that knows the outer key. Obviously, it provides no protections against an attacker that holds both the inner and outer keys.

The protections with regard to the network are the same as with the normal SRTP AES-GCM transforms. The major difference is that the double transforms are designed to work better in a group context. In such contexts, it is important to note that because these transforms are symmetric, they do not protect against attacks within the group. Any member of the group can generate valid SRTP packets for any SSRC in use by the group.

With regard to a malicious MD, the recipient can verify the integrity of the base header fields and confidentiality and integrity of the payload. The recipient has no assurance, however, of the integrity of the header extensions in the packet.

The main innovation of this transform relative to other SRTP transforms is that it allows a partly-trusted MD to decrypt, modify, and re-encrypt a packet. When this is done, the cryptographic contexts used for decryption and re-encryption MUST use different, independent master keys. If the same context is used, the nonce formation rules for SRTP will cause the same key and nonce to be used with two different plaintexts, which substantially degrades the security of AES-GCM.

In other words, from the perspective of the MD, re-encrypting packets using this protocol will involve the same cryptographic operations as if it had established independent AES-GCM crypto contexts with the sender and the receiver. If the MD doesn’t modify any header fields, then an MD that supports AES-GCM could be unused unmodified.

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:   256 bits
    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:   256 bits
    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>.

[RFC6188] McGrew, D., "The Use of AES-192 and AES-256 in Secure RTP", RFC 6188, DOI 10.17487/RFC6188, March 2011, <https://www.rfc-editor.org/info/rfc6188>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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]
Zanaty, M., Singh, V., Begen, A., and G. Mandyam, "RTP Payload Format for Flexible Forward Error Correction (FEC)", draft-ietf-payload-flexible-fec-scheme-20 (work in progress), May 2019.

[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-05 (work in progress), April 2019.

[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 (PERC)", draft-ietf-perc-private-media-framework-12 (work in progress), June 2019.

[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-10 (work in progress), July 2019.

[I-D.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-10 (work in progress), July 2019.

[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>.

[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>.

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 hop-by-hop and end-to-end operations.

```
       0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<++
    |V=2|P|X|  CC   |M|     PT      |       sequence number         | IO
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ IO
    |                           timestamp                           | IO
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ IO
    |           synchronization source (SSRC) identifier            | IO
    +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+ IO
    |            contributing source (CSRC) identifiers             | IO
    |                               ....                            | IO
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+O
    |                    RTP extension (OPTIONAL) ...               | |O
+>+>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+O
O I |                          payload  ...                         | IO
O I |                               +-------------------------------+ IO
O I |                               | RTP padding   | RTP pad count | IO
O +>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+O
O | |                    E2E authentication tag                     | |O
O | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |O
O | |                            OHB ...                            | |O
+>| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |+
| | |                    HBH authentication tag                     | ||
| | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ||
| |                                                                   ||
| +- E2E Encrypted Portion               E2E Authenticated Portion ---+|
|                                                                      |
+--- HBH Encrypted Portion               HBH Authenticated Portion ----+
```

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 P. Jones
Internet-Draft Cisco
Intended status: Standards Track D. Benham
Expires: December 7, 2019 C. Groves
Independent
June 5, 2019

A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing (PERC)

draft-ietf-perc-private-media-framework-12

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 builds upon existing security mechanisms defined for the real-time transport protocol (RTP).

Status of This Memo

Copyright Notice

Table of Contents

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 Media Distributors to each of the other participants. Media Distributors often forward only a subset of flows based on voice activity detection or other criteria. In some instances, 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. Virtualized public cloud environments have been viewed as less secure since resources are not always physically controlled by those who use them. This document defines improved security so as to lower the barrier to taking advantage of those environments.

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 RTP Control Protocol (RTCP) [RFC3550] 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.

This solution framework provides 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 (or simply 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 devices and more that are in the trusted domain for a given deployment. In the context of WebRTC [W3C.CR-webrtc-20180927], where control of a session is divided between a JavaScript application and a browser, the browser acts as the Trusted Endpoint for purposes of this framework (just as it acts as the endpoint for DTLS-SRTP [RFC5764] in one-to-one calls).

Media Distributor (MD): An RTP middlebox that forwards Endpoint media content (e.g., voice or video data) unaltered, either a subset or all of the flows at any given time, and is never allowed 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 is not limited to, having no access to RTP media plaintext and having limits on what RTP header field it can alter. The header fields that may be modified by a Media Distributor are enumerated in Section 4 of the Double cryptographic transform specification [I-D.ietf-perc-double] and chosen with respect to utility and the security considerations outlined in this document.

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 keeps the definitions and examples focused by only using the simple, most general deployment scenario.

```
                              |
               +----------+   |      +-----------------+
               | Endpoint |   |      | Call Processing |
               +----------+   |      +-----------------+
                              |
                              |
           +----------------+ |    +--------------------+
           | Key Distributor| |    | Media Distributor  |
           +----------------+ |    +--------------------+
                              |
                Trusted       |          Untrusted
                Entities      |          Entities
                              |
```

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 is not 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 are 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 also relays secured messaging between the Endpoints and the Key Distributor and acquires per-hop key information from the Key Distributor. The actual media content must not be decryptable by a Media Distributor, as it is untrusted to have access to the E2E media encryption keys. The key exchange mechanisms specified in this framework prevent the Media Distributor from gaining access to the E2E media encryption keys.

An Endpoint's ability to connect to a conference serviced by a Media Distributor does imply that the Endpoint is 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 the Endpoint (e.g., using WebRTC Identity [I-D.ietf-rtcweb-security-arch]) and determining its authorization to receive E2E and HBH 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 8) to a level equal to or better than traditional conferencing (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.

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 has 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 take measures to mitigate the adverse effects of denial of service attacks (refer to Section 8) from other entities, including from other Endpoints, to a level equal to or better than traditional conference (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 (E2E) and hop-by-hop (HBH) 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 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.

They Key Distributor needs to know which Endpoints and which Media Distributors are authorized to participate in the conference. How the Key Distributor obtains this information is outside the scope of this document. However, Key Distributors MUST reject DTLS associations with any unauthorized Endpoint or Media Distributor.

4. Framework for PERC

The purpose for this framework is to define a means through which media privacy is 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 Secure Real-time Transport Protocol (SRTP) [RFC3711], Encrypted Key Transport (EKT) [I-D.ietf-perc-srtp-ekt-diet], and DTLS-SRTP.

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 to only trusted entities to the E2E key used for authenticated end-to-end encryption. However, this framework does give a Media Distributor access to RTP headers fields and header extensions, as well as the ability to modify a certain subset of the header fields and to add or change 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 or the adjacent Endpoint for the hop between an Endpoint and a Media Distributor or peer Endpoint. An Endpoint will receive one or more E2E keys from every other Endpoint in the conference that correspond to the media flows transmitted by those other Endpoints, while HBH keys are derived from the DTLS-SRTP association with the Key Distributor. Two communicating Media Distributors use DTLS-SRTP associations directly with each other to obtain the HBH keys they will use. See Section 4.5 for more details on key exchange.

```
+-------------+                                +-------------+
|             |################################|             |
|    Media    |------------------------ *----->|    Media    |
| Distributor |<----------------------*-|------| Distributor |
|      X      |#####################*#|#|######|      Y      |
|             |                     | | |      |             |
+-------------+                     | | |      +-------------+
   #  ^ |  #          HBH Key (XY) -+ | |         #  ^ |  #
   #  | |  #           E2E Key (B) ---+ |         #  | |  #
   #  | |  #           E2E Key (A) -----+         #  | |  #
   #  | |  #                                      #  | |  #
   #  | |  #                                      #  | |  #
   #  | |  *---- HBH Key (AX)    HBH Key (YB) ----*  | |  #
   #  | |  #                                      #  | |  #
   #  *--------- E2E Key (A)      E2E Key (A) ---------*  #
   #  | *------- E2E Key (B)      E2E Key (B) -------* |  #
   #  | |  #                                      #  | |  #
   #  | v  #                                      #  | v  #
+-------------+                                +-------------+
| Endpoint A  |                                | Endpoint B  |
+-------------+                                +-------------+
```

Figure 2: E2E and HBH Keys Used for Authenticated Encryption of SRTP Packets

The Double transform [I-D.ietf-perc-double] enables Endpoints to perform encryption using both the end-to-end and hop-by-hop 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 Section 6.1 for a description of the keys used in PERC and Section 7 for diagram of how encrypted RTP packets appear on the wire.

RTCP is only 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. For this reason, Endpoints MUST NOT send confidential information via RTCP.

4.2. E2E Key Confidentiality

To ensure the confidentiality of E2E keys shared between Endpoints, Endpoints use a common Key Encryption Key (KEK) that is known only by the trusted entities in a conference. That KEK, defined in the EKT specification [I-D.ietf-perc-srtp-ekt-diet] as the EKT Key, is 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 creates an SRTP master key for E2E authenticated encryption and keep track of the E2E keys received via the Full EKT Tag for each distinct synchronization source (SSRC) in the conference so that it can properly decrypt received media. 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 is identified by its SSRC, and an Endpoint might send more than one at a time and change the mix of media flows transmitted during the life of a conference.

Thus, an Endpoint MUST maintain a list of SSRCs from received RTP flows and each SSRC's associated E2E key information. An Endpoint MUST discard old E2E keys no later than when it leaves the conference (see Section 4.5.2).

If the packet is to contain RTP header extensions, it should be noted that those are only encrypted HBH per [I-D.ietf-perc-double]. For this reason, Endpoints MUST NOT transmit confidential information via RTP header extensions.

4.4. HBH Keys and Per-hop Operations

To ensure the integrity of transmitted media packets, it is REQUIRED that every packet be authenticated hop-by-hop 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.

While Endpoints also perform HBH authentication, the ability of the Endpoints to reconstruct the original RTP header also enables the Endpoints to authenticate RTP packets E2E. This design yields flexibility to the Media Distributor to change certain RTP header values as packets are forwarded. Which values the Media Distributor can change in the RTP header are defined in [I-D.ietf-perc-double]. RTCP can only be encrypted hop-by-hop, 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 8). The use of the replay protection mechanism specified in Section 3.3.2 of [RFC3711] is REQUIRED at each hop.

If there is a need to encrypt one or more RTP header extensions hop-by-hop, the Endpoint derives an encryption key from the HBH SRTP master key to encrypt header extensions as per [RFC6904]. This still gives 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 need to decrypt and re-encrypt these encrypted header extensions. Please refer to Sections 5.1 through 5.3 of [I-D.ietf-perc-double] for procedures to perform RTP header extension encryption and decryption.

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 the DTLS-SRTP association with the Key Distributor or chosen by the Endpoint. It is then distributed to other Endpoints in a Full EKT Tag, encrypted under an EKT Key provided to the client by the Key Distributor within the DTLS channel they negotiated. Note that an Endpoint MAY create a different E2E key per media flow, where a media flow is identified by its SSRC value.

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

- The HBH keys used between two Media Distributors is derived from DTLS-SRTP procedures employed directly between them.

4.5.1. Initial Key Exchange and Key Distributor

The Media Distributor maintains a tunnel with the Key Distributor (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 enables each Endpoint to generate E2E and HBH keys and receive the KEK. At the same time, the Key Distributor securely provides 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.

```
                             +-----------+
                    KEK info |    Key    | HBH Key info to
                to Endpoints |Distributor| Endpoints & Media Distributor
                             +-----------+
                                # ^ ^ #
                                # | | #--- Tunnel
                                # | | #
   +-----------+             +-----------+             +-----------+
   | Endpoint  |   DTLS      |   Media   |   DTLS      | Endpoint  |
   |    KEK    |<------------|Distributor|------------>|    KEK    |
   |  HBH Key  | to Key Dist | HBH Keys  | to Key Dist |  HBH Key  |
   +-----------+             +-----------+             +-----------+
```

Figure 3: Exchanging Key Information Between Entities

In addition to the secure tunnel between the Media Distributor and the Key Distributor, there are two additional types of security associations utilized as a part of the key exchange as discussed in the following paragraphs. One is a DTLS-SRTP association between an Endpoint and the Key Distributor (with packets passing through the Media Distributor) and the other is a DTLS-SRTP association between peer Media Distributors.

Endpoints establish a DTLS-SRTP association over the RTP session with the Media Distributor and its media ports for the purposes of key information exchange with the Key Distributor. The Media Distributor does not terminate the DTLS signaling, but instead forwards 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.

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 KEK is conveyed by the Key Distributor over the DTLS association to Endpoints via procedures defined in EKT [I-D.ietf-perc-srtp-ekt-diet] via the EKTKey message.

The Key Distributor MUST NOT establish DTLS-SRTP associations with Endpoints without first authenticating the Media Distributor tunneling the DTLS-SRTP packets from the Endpoint.

Note that following DTLS-SRTP procedures for the [I-D.ietf-perc-double] cipher, the Endpoint generates both E2E and HBH encryption keys and salt values. Endpoints MUST either use the DTLS-SRTP generated E2E key for transmission or 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 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 is able to encrypt media end-to-end with an E2E key, sending that E2E key to other Endpoints encrypted with the KEK, and is 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 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 new EKTKey message containing the new EKT Key to each of the conference participants. Upon receipt of the new EKT Key, the Endpoint MUST create a new SRTP master key and prepare to send that key inside a Full EKT Field using the new EKT Key as per Section 4.5 of [I-D.ietf-perc-srtp-ekt-diet]. In order to allow time for all Endpoints in the conference to receive the new keys, the sender should follow the recommendations in Section 4.7 of [I-D.ietf-perc-srtp-ekt-diet]. On receiving a new EKT Key, Endpoints MUST be prepared to decrypt EKT tags using the new key. The EKT SPI field is used to differentiate between tags encrypted with the old and new keys.

After re-keying, an Endpoint SHOULD retain prior SRTP master keys and EKT Key for a period of time sufficient for the purpose of ensuring it can decrypt late-arriving or out-of-order packets or packets sent by other Endpoints that used the prior keys for a period of time after re-keying began. An Endpoint MAY retain old keys until the end of the conference.

Endpoints MAY follow the procedures in section 5.2 of [RFC5764] to renegotiate 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] as one possible method.

Endpoints MAY change the E2E encryption key used at any time. An Endpoint MUST generate a new E2E encryption key whenever it receives a new EKT Key. After switching to a new key, the new key is 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 critical requirements are that an Endpoint can verify it is connected to the correct Key Distributor for the conference and the Key Distributor can verify the Endpoint is the correct Endpoint 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] is 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 Session Description Protocol (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 [RFC8224] for the benefit of forwarding the message to other entities by the Focus [RFC4353]. Other entities can 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 verify 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 an Endpoint in a manner that can be authenticated as being an authorized Key Distributor for this conference.

5.3. Conference Identification

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

6. PERC Keys

This section describes the various keys employed by PERC.

6.1. Key Inventory and Management Considerations

This section summarizes the several different keys used in the PERC framework, 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 4 below.

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

Figure 4: Key Inventory

While the number of key types is very small, it should be understood that the actual number of distinct keys can be large as the conference grows in size.

As an example, with 1,000 participants in a conference, there would be at least 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 that are associated with a new EKT SPI. Endpoints might retain old keys for a period of time to ensure they can properly decrypt late-arriving or out-of-order packets, which means the number of keys held during that period of time might substantially more.

A more detailed explanation of each of the keys follows.

6.2. DTLS-SRTP Exchange Yields HBH Keys

The first set of keys acquired are for hop-by-hop encryption and decryption. Per the Double [I-D.ietf-perc-double] procedures, the Endpoint performs DTLS-SRTP exchange with the Key Distributor and receives a key that is, in fact, "double" the size that is needed. The end-to-end part is the first half of the key, so the Endpoint discards that information when generating its own key. The second half of the key material is for hop-by-hop operations, so that half of the key (corresponding to the least significant bits) is assigned internally as the HBH key.

The Key Distributor informs the Media Distributor of the HBH key. Specifically, the Key Distributor sends the least significant bits corresponding to the half of the keying material determined through

DTLS-SRTP with the Endpoint to the Media Distributor. A salt value is generated along with the HBH key. The salt is also longer than needed for hop-by-hop operations, thus only the least significant bits of the required length (half of the generated salt material) are sent to the Media Distributor. One way to transmit this key and salt information is via the tunnel protocol defined in [I-D.ietf-perc-dtls-tunnel].

No two Endpoints 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.

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

6.3. The Key Distributor Transmits the KEK (EKT Key)

Via the aforementioned DTLS-SRTP association, the Key Distributor sends the Endpoint the KEK (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) allows 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 is used to re-key the entire conference. Each key is identified by a "Security Parameter Index" (SPI) value. Endpoints MUST 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 Full EKT Tag 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 Tag data as per [I-D.ietf-perc-srtp-ekt-diet]).

The Media Distributor is not given the KEK.

6.4. 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 E2E SRTP master key. While the DTLS-SRTP-derived SRTP master key can be used initially, the Endpoint might choose to change the SRTP master key periodically and MUST change the SRTP master key as a result of the EKT key changing.

A 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 does 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 transmits its E2E key to other Endpoints in the conference by periodically including it in SRTP packets in a Full EKT Tag. When placed in the Full EKT Tag, it is encrypted using the EKT Key provided by the Key Distributor. The master salt is not transmitted, though, since all Endpoints receive the same master salt via the EKTKey message from the Key Distributor. The recommended frequency with which an Endpoint transmits its SRTP master key is specified in [I-D.ietf-perc-srtp-ekt-diet].

6.5. Summary of Key Types and Entity Possession

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. Since HBH keys are derived from a DTLS-SRTP association, there is at most one HBH key per Endpoint, (The only exception is where the DTLS-SRTP association might be rekeyed per Section 5.2 of [RFC5764] and a new key is created to replace the former key.)

Each Endpoint generates its own E2E key (SRTP master key), thus there is a distinct E2E key per endpoint. This key is transmitted (encrypted) via the Full EKT Tag to other Endpoints. Endpoints that receive media from a given transmitting Endpoint gain knowledge of the transmitter's E2E key via the Full EKT Tag.

To summarize the various keys and which entity is in possession of a given key, refer to Figure 5.

Key / Entity	Endpoint A	MD X	MD Y	Endpoint B
KEK (EKT Key)	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 5: Keys Types and Entity Possession

7. Encrypted Media Packet Format

Figure 6 presents a complete picture of what an encrypted media packet per this framework looks like when transmitted over the wire. The packet format shown is encrypted using the Double cryptographic transform with an EKT Tag appended to the end.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<++
    |V=2|P|X|  CC   |M|     PT      |       sequence number         | IO
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ IO
    |                           timestamp                           | IO
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ IO
    |           synchronization source (SSRC) identifier            | IO
    +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+ IO
    |            contributing source (CSRC) identifiers             | IO
    |                               ....                            | IO
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+O
    |                   RTP extension (OPTIONAL) ...                | |O
+>+>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+O
O I |                          payload  ...                         | IO
O I |                               +-------------------------------+ IO
O I |                               | RTP padding   | RTP pad count | IO
O +>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+O
O | |                   E2E authentication tag                      | |O
O | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |O
O | |                            OHB ...                            | |O
+>| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |+
| | |                   HBH authentication tag                      | ||
| | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ||
| | |   EKT Tag ...   | R                                             ||
| | +-+-+-+-+-+-+-+-+-+ |                                             ||
| |                     +- Neither encrypted nor authenticated;       ||
| |                        appended after Double is performed         ||
| |                                                                   ||
| |                                                                   ||
| +- E2E Encrypted Portion               E2E Authenticated Portion ---+|
|                                                                      |
+--- HBH Encrypted Portion               HBH Authenticated Portion ----+

    I = Inner (E2E) encryption / authentication
    O = Outer (HBH) encryption / authentication
```

Figure 6: Encrypted Media Packet Format

8. Security Considerations

8.1. Third Party Attacks

Third party attacks are attacks attempted by an adversary that is not supposed to have access to keying material or is otherwise not an authorized participant in the conference.

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

Off-path attackers could try connecting to different PERC entities to send specifically crafted packets with an aim of forcing the receiver to forward or render bogus media packets. Endpoints and Media Distributors mitigate such an attack by performing hop-by-hop authentication and discarding packets that fail authentication.

Another attack vector is a third party claiming to be a Media Distributor, fooling Endpoints into sending packets to the false Media Distributor instead of the correct one. The deceived sending Endpoints could incorrectly assume their packets have been delivered to Endpoints when they in fact have not. While this attack is possible, the result is a simple denial of service with no leakage of confidential information, since the false Media Distributor would not have access to either HBH or E2E encryption keys.

A third party could cause a denial-of-service by transmitting many bogus or replayed packets toward receiving devices that ultimately degrade conference or device performance. Therefore, implementations might wish to devise mechanisms to safeguard against such illegitimate packets, such as utilizing rate-limiting or performing basic sanity-checks on packets (e.g., looking at packet length or expected sequence number ranges) before performing more expensive decryption operations.

Use of mutual DTLS authentication (as required by DTLS-SRTP) also helps to prevent a denial-of-service attack by preventing a false Endpoint or false Media Distributor from successfully participating as a perceived valid media sender that could otherwise carry out an on-path attack. When mutual authentication fails, a receiving Endpoint would know that it could safely discard media packets received from the Endpoint without inspection.

8.2. Media Distributor Attacks

A malicious or compromised Media Distributor can attack the session in a number of possible ways.

8.2.1. Denial of service

A simple form of attack is discarding received packets that should be forwarded. This solution framework does not introduce any mitigation for Media Distributors that fail to forward media packets.

Another form of attack is modifying received packets before forwarding. With this solution framework, any modification of the end-to-end authenticated data results 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 along with a Full EKT Tag. Since such a message would trigger the receiver to form a new cryptographic context, the Media Distributor can attempt to consume the receiving Endpoint's resources. While E2E authentication would fail and the cryptographic context would be destroyed, the key derivation operation would nonetheless consume some computational resources. While resource consumption attacks cannot be mitigated entirely, rate-limiting packets might help reduce the impact of such attacks.

8.2.2. Replay Attack

A replay attack is when an already received packet 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.

A replay attack is mitigated by the requirement to implement replay protection as described in Section 3.3.2 of [RFC3711]. End-to-end replay protection MUST be provided for the duration of the conference.

8.2.3. Delayed Playout Attack

A delayed playout attack is one where media is received and held by a Media Distributor and then forwarded to Endpoints at a later point in time.

This attack is possible even if E2E replay protection is in place. Because the Media Distributor is allowed to select a subset 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.

While the Media Distributor can purposely stop forwarding media flows, it can also select an arbitrary starting point to resume forwarding those media flow, perhaps forwarding old packets rather than current packets. As a consequence, what the media source sent can be substantially delayed at the receiver with the receiver believing that newly arriving packets are delayed only by transport delay when the packets may actually be minutes or hours old.

While this attack cannot be eliminated entirely, its effectiveness can be reduced by re-keying the conference periodically since significantly-delayed media encrypted with expired keys would not be decrypted by Endpoints.

8.2.4. Splicing Attack

A splicing attack is an attack where a Media Distributor receiving multiple media sources splices one media stream into the other. If the Media Distributor were 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. By including the SSRC in the integrity check for each packet, both HBH and E2E, PERC prevents splicing attacks.

8.2.5. RTCP Attacks

PERC does not provide end-to-end protection of RTCP messages. This allows a compromised Media Distributor to impact any message that might be transmitted via RTCP, including media statistics, picture requests, or loss indication. It is also possible for a compromised Media Distributor to forge requests, such as requests to the Endpoint to send a new picture. Such requests can consume significant bandwidth and impair conference performance.

8.3. Key Distributor Attacks

As stated in Section 3.2.2, the Key Distributor needs to be secured since exploiting the Key Server can allow an adversary to gain access to the keying material for one or more conferences. Having access to that keying material would then allow the adversary to decrypt media sent from any Endpoint in the conference.

As a first line of defense, the Key Distributor authenticates every security association, both associations with Endpoints and Media Distributors. The Key Distributor knows which entities are authorized to have access to which keys and inspection of certificates will substantially reduce the risk of providing keys to an adversary.

Both physical and network access to the Key Distributor should be severely restricted. This may be more difficult to achieve when the Key Distributor is embedded within and Endpoint, for example.

Nonetheless, consideration should be given to shielding the Key Distributor from unauthorized access or any access that is not strictly necessary for the support of an ongoing conference.

Consideration should be given to whether access to the keying material will be needed beyond the conclusion of a conference. If not needed, the Key Distributor's policy should be to destroy the keying material once the conference concludes or when keying material changes during the course of the conference. If keying material is needed beyond the lifetime of the conference, further consideration should be given to protecting keying material from future exposure. While it might be obvious, it is worth stating to avoid any doubt that if an adversary were to record the media packets transmitted during a conference and then gain unauthorized access to the keying material left unsecured on the Key Distributor even years later, the adversary could decrypt the content every packet transmitted during the conference.

8.4. Endpoint Attacks

A Trusted Endpoint is so named because conference confidentiality relies heavily on the security and integrity of the Endpoint. If an adversary successfully exploits a vulnerability in an Endpoint, it might be possible for the adversary to obtain all of the keying material used in the conference. With that keying material, an adversary could decrypt any of the media flows received from any other Endpoint, either in real-time or at a later point in time (assuming the adversary makes a copy of the media packets).

Additionally, if an adversary successfully exploits an Endpoint, the adversary could inject media into the conference. One way an adversary could do that is by manipulating the RTP or SRTP software to transmit whatever media the adversary wishes to send. This could involve re-use of the Endpoint's SSRC, a new SSRC, or the SSRC value of an existing endpoint. This is made possible since all conference participants share the same KEK. Only a single SRTP cipher suite defined provides source authentication properties that allow an endpoint to cryptographically assert that it sent a particular E2E protected packet (namely, TESLA [RFC4383]), and its usage is presently not defined for PERC. The suite defined in PERC only allows an Endpoint to determine that whoever sent a packet had received the KEK.

However, attacks on the endpoint are not limited to the PERC-specific software within the Endpoint. An attacker could inject media or record media by manipulating the software that sits between the PERC-enabled application and the hardware microphone of video camera, for example. Likewise, an attacker could potentially access confidential media by accessing memory, cache, disk storage, etc. if the endpoint is no secured.

9. IANA Considerations

There are no IANA considerations for this document.

10. Acknowledgments

The authors would like to thank Mo Zanaty, Christian Oien, and Richard Barnes 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. We would also like to acknowledge John Mattsson, Mats Naslund, and Magnus Westerlund for providing some of the text in the document, including much of the original text in the security considerations section.

11. References

11.1. Normative References

[I-D.ietf-perc-double]
Jennings, C., Jones, P., Barnes, R., and A. Roach, "SRTP Double Encryption Procedures", draft-ietf-perc-double-10 (work in progress), October 2018.

[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-09 (work in progress), October 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>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.2. Informative References

[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-05 (work in progress), April 2019.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-18 (work in progress), February 2019.

[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>.

[RFC4383] Baugher, M. and E. Carrara, "The Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Secure Real-time Transport Protocol (SRTP)", RFC 4383, DOI 10.17487/RFC4383, February 2006, <https://www.rfc-editor.org/info/rfc4383>.

[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>.

[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>.

[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>.

[RFC8224] Peterson, J., Jennings, C., Rescorla, E., and C. Wendt, "Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 8224, DOI 10.17487/RFC8224, February 2018, <https://www.rfc-editor.org/info/rfc8224>.

[W3C.CR-webrtc-20180927] Bergkvist, A., Burnett, D., Jennings, C., Narayanan, A., Aboba, B., Brandstetter, T., and J. Bruaroey, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium CR CR-webrtc-20180927, September 2018, <https://www.w3.org/TR/2018/CR-webrtc-20180927>.

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
Independent

Email: dabenham@gmail.com

Christian Groves
Independent
Melbourne
Australia

Email: cngroves.std@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

C. Jennings
Cisco Systems
J. Mattsson
Ericsson AB
D. McGrew
Cisco Systems
D. Wing

F. Andreason
Cisco Systems

July 8, 2019

Encrypted Key Transport for DTLS and Secure RTP
draft-ietf-perc-srtp-ekt-diet-10

Abstract

Encrypted Key Transport (EKT) is an extension to DTLS (Datagram Transport Layer Security) 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

Copyright Notice

Table of Contents

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.

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 within a session without coordinating with other entities via external signaling or other external means.

EKT extends DTLS and SRTP to enable a common key encryption key (called an EKTKey) to be distributed to all endpoints, so that each endpoint can securely send 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 cannot 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 and 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 distributing the security parameters that an endpoint needs to associate with a given SSRC in order to decrypt SRTP packets from that sender.

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. This document defines how EKT works with the DTLS-SRTP approach to key establishment, by using keys derived from the DTLS-SRTP handshake to encipher the EKTKey in addition to the SRTP media.

2. Overview

This specification defines a way for the server in a DTLS-SRTP negotiation, see Section 5, to provide an EKTKey to the client during the DTLS handshake. The EKTKey thus obtained can be used to encrypt the SRTP master key that is used to encrypt the media sent by the endpoint. This specification also defines a way to send the encrypted SRTP master key (with the EKTKey) along with the SRTP packet, see Section 4. Endpoints that receive this and know the EKTKey can use the EKTKey to decrypt the SRTP master key which can then be used to decrypt the SRTP packet.

One way to use this is described in the architecture defined by [I-D.ietf-perc-private-media-framework]. Each participant in the conference forms a DTLS-SRTP connection to a common key distributor that distributes the same EKTKey to all the endpoints. Then each endpoint picks its own SRTP master key for the media they send. When sending media, the endpoint also includes the SRTP master key encrypted with the EKTKey in the SRTP packet. 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 field, called EKTField, is added to the SRTP packets. The EKTField appears at the end of the SRTP packet. It appears after the optional authentication tag if one is present, otherwise the EKTField 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.

This document defines the use of EKT with SRTP. Its use with SRTCP would be similar, but is reserved for a future specification. SRTP is preferred for transmitting key material because it shares fate with the transmitted media, because SRTP rekeying can occur without concern for RTCP transmission limits, and because it avoids the need for SRTCP compound packets with RTP translators and mixers.

4.1. EKTField Formats

The EKTField uses the format defined in Figure 1 for the FullEKTField and ShortEKTField.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                                                               :
 :                        EKT Ciphertext                         :
 :                                                               :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Security Parameter Index    | Length                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |0 0 0 0 0 0 1 0|
 +-+-+-+-+-+-+-+-+
```

Figure 1: FullEKTField format

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 0|
+-+-+-+-+-+-+-+-+
```

Figure 2: ShortEKTField 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)
Epoch = 2BYTE
SPI = 2BYTE

FullEKTField = EKTCiphertext SPI Epoch 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 and its length, 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 is the SSRC for this SRTP source. The length of this field is 32 bits. The SSRC value in the EKT tag MUST be the same as the one in the header of the SRTP packet to which the tag is appended.

Rollover Counter (ROC): On the sender side, this is set to the current value of the SRTP rollover counter in the SRTP/SRTCP 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 EKTKey and other parameters for the receiver to use when processing the packet, within a given conference. The length of this field is 16 bits, representing a two-byte integer in network byte order. The parameters identified by this field are:

- The EKT cipher used to process the packet.

- The EKTKey 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. (Recall that the SRTP master salt is used in the formation of IVs / nonces.)

Epoch: This field indicates how many SRTP keys have been sent for this SSRC under the current EKTKey, prior to the current key, as a two-byte integer in network byte order. It starts at zero at the beginning of a session and resets to zero whenever the EKTKey is changed (i.e., when a new SPI appears). The epoch for an SSRC increments by one every time the sender transmits a new key. The recipient of a FullEKTField MUST reject any future FullEKTField for this SPI and SSRC that has an equal or lower epoch value to an epoch already seen.

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 messages types other than the ShortEKTField have a length as second from the last element. This is the length in octets (in network byte order) of either the FullEKTField/ ExtensionEKTField including this length field and the following EKT Message Type.

Message Type: The last byte is used to indicate the type of the EKTField. This MUST be 2 for the FullEKTField format and 0 in ShortEKTField format. If a received EKT tag has an unknown message type, then the receiver MUST discard the whole EKT tag.

4.2. SPIs and EKT Parameter Sets

The SPI field identifies the parameters for how the EKT tag should be processed:

- o The EKTKey and EKT cipher used to process the packet.

- o 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. The association of a given parameter set with a given SPI value is configured by some other protocol, e.g., the DTLS-SRTP extension defined in Section 5.

4.3. 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. If a participant deletes an EKT parameter set (e.g., because of space limitations, then it will be unable to process Full EKT Tags containing updated media keys, and thus unable to receive media from a particpant that has changed its media key.

Either the FullEKTField or ShortEKTField is appended at the tail end of all SRTP packets. The decision on which to send when is specified in Section 4.6.

4.3.1. Outbound Processing

See Section 4.6 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 MUST 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 EKTCipher 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 appended to the end of the SRTP packet, after the encrypted payload.

When a packet is sent with the ShortEKTField, the ShortEKFField is simply appended to the packet.

Outbound packets SHOULD continue to use the old SRTP Master Key for 250 ms after sending any new key in a FullEKTField value. This gives all the receivers in the system time to get the new key before they start receiving media encrypted with the new key. (The specific value of 250ms is chosen to represent a reasonable upper bound on the amount of latency and jitter that is tolerable in a real-time context.)

4.3.2. Inbound Processing

When receiving a packet on a RTP stream, the following steps are applied for each SRTP 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 EKTField 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 at the end of the packet and thus the type is placed at the very end of the packet.

2. The Security Parameter Index (SPI) field is used to find the right EKT parameter set to be used for 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 the SRTP Master Salt.

3. The EKTCiphertext is authenticated and 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 EKT processing MUST be aborted. The receiver SHOULD discard the whole UDP packet.

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 received, then all the information from this EKTPlaintext MUST be discarded and the following steps in this list are skipped.

6. The SRTP Master Key, ROC, and SRTP Master Salt from the previous steps 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.

  * Unless the transform specifies other acceptable key lengths, the length of the SRTP Master Key MUST be the same as the master key length for the SRTP transform in use. If this is not the case, then the receiver MUST abort EKT processing and SHOULD discared the whole UDP packet.

  * If the length of the SRTP Master Key is less than the master key length for the SRTP transform in use, and the transform specifies that this length is acceptable, then the SRTP Master Key value is used to replace the first bytes in the existing master key. The other bytes remain the same as in the old key. For example, the Double GCM transform [I-D.ietf-perc-double] allows replacement of the first, "end to end" half of the master key.

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

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. SRTP senders and receivers MAY cache an EKTCiphertext value to optimize processing in cases where the master key hasn’t changed. Instead of encrypting and decrypting, senders can simply copy the pre-computed value and receivers can compare a received EKTCiphertext to the known value.

Section 4.3.1 recommends that SRTP senders continue using an old key for some time after sending a new key in an EKT tag. Receivers that wish to avoid packet loss due to decryption failures MAY perform trial decryption with both the old key and the new key, keeping the result of whichever decryption succeeds. Note that this approach is only compatible with SRTP transforms that include integrity protection.

When receiving a new EKTKey, implementations need to use the ekt_ttl field (see Section 5.2.2) 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 key has been used to encrypt data to ensure it does not exceed the T value for that cipher (see Section 4.4). 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 renegotiate 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. The cipher used for a given EKTCiphertext value is negotiated using the supported_ekt_ciphers and indicated with the SPI value in the FullEKTField.

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. AES Key Wrap

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. Timing and Reliability Consideration

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

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 FullEKTField be transmitted. If the sender does not send a FullEKTField in its initial packets and receivers have not otherwise been provisioned with a decryption key, then decryption will fail and SRTP packets will be dropped until the receiver receives a FullEKTField from the sender.

Rekey:
: By sending EKT tag 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 FullEKTField. 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 EKTKey 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.

In cases where the DTLS termination point is more trusted than the media relay, the protection that DTLS affords to EKT key material can allow EKT keys to be tunneled through an untrusted relay such as a centralized conference bridge. For more details, see [I-D.ietf-perc-private-media-framework].

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.

Figure 4 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.) 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.

```
      Client                                               Server

      ClientHello
       + use_srtp
       + supported_ekt_ciphers
                               -------->

                                                       ServerHello
                                             {EncryptedExtensions}
                                                        + use_srtp
                                           + supported_ekt_ciphers
                                                    {... Finished}
                               <--------

      {... Finished}           -------->

                                                             [Ack]
                               <--------                  [EKTKey]

      [Ack]                    -------->

      |SRTP packets|           <------->            |SRTP packets|
      + <EKT tags>                                    + <EKT tags>


      {} Messages protected using DTLS handshake keys

      [] Messages protected using DTLS application traffic keys

      <> Messages protected using the EKTKey and EKT cipher

      || Messages protected using the SRTP Master Key sent in
         a Full EKT Tag
```

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 document allow the DTLS server to set a common EKTKey for all participants. Each endpoint can then use EKT tags encrypted with that common key to inform other endpoint of the keys it uses to protect SRTP packets. This avoids the need for many individual DTLS handshakes among the endpoints, at the cost of preventing endpoints from directly authenticating one another.

```
        Client A                 Server                 Client B

            <----DTLS Handshake---->
            <--------EKTKey---------
                                    <----DTLS Handshake---->
                                    ---------EKTKey-------->

            -------------SRTP Packet + EKT Tag------------->
            <------------SRTP Packet + EKT Tag--------------
```

5.2.1. Negotiating an EKTCipher

To indicate its support for EKT, a DTLS-SRTP client includes in its ClientHello an extension of type supported_ekt_ciphers listing the ciphers used for EKT by 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 [RFC8446]:

```
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 EKTCipher, the server MUST provide to the client a key to be used when encrypting and decrypting EKTCiphertext values. EKTKeys 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 EKTKey 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 EKTKey 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 EKTKey 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 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].

EKT MAY be used with versions of DTLS prior to 1.3. In such cases, the Ack message is still used to provide reliability. Thus, DTLS implementations supporting EKT with DTLS pre-1.3 will need to have explicit affordances for sending the Ack message in response to an EKTKey message, and for verifying that an Ack message was received. The retransmission rules for both sides are the same as in DTLS 1.3.

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 the key management protocol that is used to establish the EKTKey, e.g., the DTLS-SRTP extension defined in this document.

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. This mitigates the impact of the cut-and-paste attacks that arise due to the lack of a cryptographic binding between the EKT tag and the rest of the SRTP packet. SRTP tags can only be cut-and-pasted within the stream of packets sent by a given RTP endpoint; an attacker cannot "cross the streams" and use an EKT tag from one SSRC to reset the key for another SSRC. The epoch field in the FullEKTField also prevents an attacker from rolling back to a previous key.

An attacker could send packets containing a FullEKTField, in an attempt to consume additional CPU resources of the receiving system by causing the receiving system to decrypt the EKT ciphertext and detect an authentication failure. In some cases, caching the previous values of the Ciphertext as described in Section 4.3.2 helps mitigate this issue.

In a similar vein, EKT has no replay protection, so an attacker could implant improper keys in receivers by capturing EKTCiphertext values encrypted with a given EKTKey and replaying them in a different context, e.g., from a different sender. When the underlying SRTP transform provides integrity protection, this attack will just result in packet loss. If it does not, then it will result in random data being fed to RTP payload processing. An attacker that is in a position to mount these attacks, however, could achieve the same effects more easily without attacking EKT.

The key encryption keys distributed with EKTKey messages are group shared symmetric keys, which means they do not provide protection within the group. Group members can impersonate each other; for example, any group member can generate an EKT tag for any SSRC. The entity that distributes EKTKeys can decrypt any keys distributed using EKT, and thus any media protected with those keys.

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 FullEKTField values using the same EKTKey. In addition, the EKTKey MUST NOT be used beyond the lifetime provided by the TTL described in Section 5.2.

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 MUST 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
Unallocated	3-254	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]. 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. Allocated values MUST be in the range of 0 to 254.

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

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	0	RFCAAAA
AESKW256	1	RFCAAAA
Unallocated	2-254	
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 0 to 254.

7.3. TLS Extensions

IANA is requested to add supported_ekt_ciphers as a new extension name to the "TLS ExtensionType Values" table of the "Transport Layer Security (TLS) Extensions" registry:

Value: [TBD-at-Registration]
Extension Name: supported_ekt_ciphers
TLS 1.3: CH, SH
Recommended: Y
Reference: RFCAAAA

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

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:

Value: [TBD-at-Registration]
Description: ekt_key
DTLS-OK: Y
Reference: RFCAAAA
Comment:

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

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>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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-10 (work in progress), October 2018.

[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 (PERC)", draft-ietf-perc-private-media-framework-12 (work in progress), June 2019.

[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-dtls13-31 (work in progress), March 2019.

[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>.

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

Email: dwing-ietf@fuggles.com

Flemming Andreason
Cisco Systems

Email: fandreas@cisco.com