Double Encryption Procedures for the Secure Real-Time Transport Protocol
(SRTP)

Abstract

In some conferencing scenarios, it is desirable for an intermediary
to be able to manipulate some parameters in Real-time Transport
Protocol (RTP) packets, while still providing strong end-to-end
security guarantees. This document defines a cryptographic transform
for the Secure Real-time Transport 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 (MD) 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 an RTP packet to be changed by the MD.
   At the same time, a separate cryptographic key provides integrity and
   optional confidentiality for the media flowing between the MD and the
   endpoints. The framework document [PRIVATE-MEDIA-FRAMEWORK]
   describes this concept in more detail.

   This specification defines a transform for SRTP that uses 1) the AES
   Galois/Counter Mode (AES-GCM) algorithm [RFC7714] to provide
   encryption and integrity for an RTP packet for the end-to-end
   cryptographic key and 2) a hop-by-hop cryptographic encryption and
   integrity between the endpoint and the MD. The MD decrypts and
   checks integrity of the hop-by-hop security. The MD 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 MD
   changed using the information in the Original Header Block before
   decrypting and checking the end-to-end integrity.

One can think of the double transform as a normal SRTP transform for encrypting the RTP in a way such that 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 (MD): 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 [PRIVATE-MEDIA-FRAMEWORK]).

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

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

Original Header Block (OHB): An octet string that contains the original values from the RTP header that might have been changed by an MD.

3. Cryptographic Context

This specification uses a cryptographic context with two parts:

* An inner (end-to-end) part that is used by endpoints that originate and consume media to ensure the integrity of media end-to-end, and

* An outer (hop-by-hop) part that is used between endpoints and MDs to ensure the integrity of media over a single hop and to enable an MD 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 algorithms is AES-GCM. Other combinations of SRTP ciphers that support the procedures in this document can be added to the IANA registry.

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

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

* Assign the key and salt values generated for the inner (end-to-end) algorithm to the first half of the key and the first half of the salt for the double algorithm.

*   Assign the key and salt values for the outer (hop-by-hop)
    algorithm to the second half of the key and second half of the
    salt for the double algorithm. The first half of the key is
    referred to as the inner key while the second half is referred to
    as the outer key. When a key is used by a cryptographic
    algorithm, the salt that is used is the part of the salt generated
    with that key.

*   the synchronization source (SSRC) is the same for both the inner
    and outer algorithms as it cannot be changed.

*   The sequence number (SEQ) and rollover counter (ROC) are tracked
    independently for the inner and outer algorithms.

If the MD is to be able to modify header fields but not decrypt the
payload, then it must have a cryptographic key for the outer
algorithm but not the inner (end-to-end) algorithm. This document
does not define how the MD should be provisioned with this
information. One possible way to provide keying material for the
outer (hop-by-hop) algorithm is to use [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 pseudorandom 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_double_n(k_master, x)" represents the AES_CM PRF Key
Derivation Function (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.
The term "inner(k_master)" represents the first half of the key;
"outer(k_master)" represents the second half of the key.

4. Original Header Block

The 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 additional authenticated data (AAD) value
to the inner transform.

The OHB can reflect modifications to the following fields in an RTP
header: the payload type (PT), the SEQ, 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 end-to-end 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|
+-+-+-+-+-+-+-+-+
```

* P: PT is present

* Q: SEQ is present

* M: Marker bit is present

* B: Value of marker bit

* 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 protects the integrity of a version of the SRTP header with extensions truncated. Omitting extensions from the inner integrity check means that they can be modified by an MD 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.

An MD 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

An endpoint encrypts a packet by using the inner (end-to-end) cryptographic key and then 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 * CSRC count (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 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 Encrypted Key Transport (EKT) [EKT-SRTP], the EKTField comes after the SRTP packet, exactly like using EKT with any other SRTP transform.

5.2. Relaying a Packet

The MD 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 inner (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 MD 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
outer (hop-by-hop) cryptographic key before transmitting using the
following steps:

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 that are allowed to be
    changed, i.e., PT, SEQ, and M. The MD 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 the MD 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 MD 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 MD 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
steps 1 and 4 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.

An MD 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 MD 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 using the following steps:

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 MD,
    the synthetic packet has the original sequence number. If the
    integrity check does not pass, discard the packet.

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

*   The PT from the outer SRTP packet is used for normal matching to
    Session Description Protocol (SDP) and codec selection.

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

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

If 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

MDs sometimes interact with RTP media packets sent by endpoints,
e.g., to provide recovery or receive commands via dual-tone multi-
frequency (DTMF) signaling. 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 MD 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 an MD 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 the double transform, 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 like a packet in repair
mode (i.e., with only the hop-by-hop key).

If the MD were to cache the inner, E2E-encrypted payload and
retransmit it with an RTX original sequence number 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 the double transform, the processing at
the sender and receiver is the same as when using RED with any other
SRTP transform.

The main difference between the double transform and any other
transform is that in an intermediated environment, usage of RED must

be end-to-end. An MD cannot synthesize RED packets, because it lacks
access to the plaintext media payloads that are combined to form a
RED payload.

Note that Flexible Forward Error Correction (Flex FEC) may often
provide similar or better repair capabilities compared to RED. For
most applications, Flex FEC is a better choice than RED; in
particular, Flex FEC has modes in which the MD can synthesize
recovery packets.

7.3. Forward Error Correction (FEC)

When using Flex FEC [RFC8627] with the double transform, 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 MD but, at the
same time, allows the MD 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 [WEBRTC-FEC] for repair of video is Flex
FEC [RFC8627]. Note that for interoperability with WebRTC,
[WEBRTC-FEC] recommends not using additional FEC-only "m=" lines in
SDP for the repair packets.

7.4. DTMF

When DTMF is sent using the mechanism in [RFC4733], it is end-to-end
encrypted; the relay cannot 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 algorithms 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; thus, it is 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 were 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 were 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: a 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. This property allows the use of an MD that
   supports AES-GCM but does not modify any header fields, without
   requiring any modification to the MD.

10. IANA Considerations

10.1. DTLS-SRTP

   IANA has added the following protection profiles to the "DTLS-SRTP
   Protection Profiles" registry defined in [RFC5764].

   +---------+--+-----------+
   | Value | Profile | Reference |
   +=========+==+===========+
   | {0x00, | DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM | RFC 8723 |

```
| 0x09} |                                                        |          |
+-------+----------------------------------------------------+----------+
|{0x00, | DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM           | RFC 8723 |
| 0x0A} |                                                    |          |
+-------+----------------------------------------------------+----------+
```

Table 1: Updates to the DTLS-SRTP Protection Profiles Registry

The SRTP transform parameters for each of these protection profiles
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      |
+-----------------------+--------------------------------------+
```

Table 2: SRTP Transform Parameters for
DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM

```
+--------------------------------------------------------------+
| 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      |
+-----------------------+--------------------------------------+
```

Table 3: SRTP Transform Parameters for DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM

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. References

11.1. Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119,
            DOI 10.17487/RFC2119, March 1997,
            <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>.

11.2. Informative References

[DTLS-TUNNEL]
            Jones, P., Ellenbogen, P., and N. Ohlmeier, "DTLS Tunnel
            between a Media Distributor and Key Distributor to
            Facilitate Key Exchange", Work in Progress, Internet-

                    Draft, draft-ietf-perc-dtls-tunnel-06, 16 October 2019,
                    <https://tools.ietf.org/html/draft-ietf-perc-dtls-tunnel-
                    06>.

     [EKT-SRTP]   Jennings, C., Mattsson, J., McGrew, D., Wing, D., and F.
                    Andreasen, "Encrypted Key Transport for DTLS and Secure
                    RTP", Work in Progress, Internet-Draft, draft-ietf-perc-
                    srtp-ekt-diet-10, 8 July 2019,
                    <https://tools.ietf.org/html/draft-ietf-perc-srtp-ekt-
                    diet-10>.

     [PRIVATE-MEDIA-FRAMEWORK]
                    Jones, P., Benham, D., and C. Groves, "A Solution
                    Framework for Private Media in Privacy Enhanced RTP
                    Conferencing (PERC)", Work in Progress, Internet-Draft,
                    draft-ietf-perc-private-media-framework-12, 5 June 2019,
                    <https://tools.ietf.org/html/draft-ietf-perc-private-
                    media-framework-12>.

     [RFC2198]   Perkins, C., Kouvelas, I., Hodson, O., Hardman, V.,
                    Handley, M., Bolot, J.C., 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>.

     [RFC8627]   Zanaty, M., Singh, V., Begen, A., and G. Mandyam, "RTP
                    Payload Format for Flexible Forward Error Correction
                    (FEC)", RFC 8627, DOI 10.17487/RFC8627, July 2019,
                    <https://www.rfc-editor.org/info/rfc8627>.

     [WEBRTC-FEC]
                    Uberti, J., "WebRTC Forward Error Correction
                    Requirements", Work in Progress, Internet-Draft, draft-
                    ietf-rtcweb-fec-10, 16 July 2019,
                    <https://tools.ietf.org/html/draft-ietf-rtcweb-fec-10>.

Appendix A. Encryption Overview

   The following figures show 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         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  RTP extension (OPTIONAL) ...                  |
+>+>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
0 I |                         payload  ...                      |
0 I |                               +-------------------------------+
0 I |                               | RTP padding   | RTP pad count |
0 +>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
0   | |                    E2E authentication tag                   |
0   | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
0   | |                           OHB ...                           |
+>  | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   | |                    HBH authentication tag                   |
|   | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   |
|   +- E2E Encrypted Portion
|
+--- HBH Encrypted Portion

 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         | I O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ I O
|                           timestamp                           | I O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ I O
|           synchronization source (SSRC) identifier            | I O
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+ I O
|            contributing source (CSRC) identifiers             | I O
|                             ....                              | I O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+ O
|                  RTP extension (OPTIONAL) ...                  | | O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+ O
|                          payload  ...                         | I O
|                               +-------------------------------+ I O
|                               | RTP padding   | RTP pad count | I O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+ O
|                     E2E authentication tag                    | | O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | O
|                           OHB ...                             | | O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |<+
|                     HBH authentication tag                    | | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | |
                                                                 | |
                        E2E Authenticated Portion ---+ |
```

```
                                             |
                      HBH Authenticated Portion -----+
```

Acknowledgments

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

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