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

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

March 5, 2018

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

Abstract

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

Status of This Memo

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 from the sending endpoint to the receiving endpoint that can encrypt and authenticate the media end-to-end while still allowing certain RTP header information to be changed by the Media Distributor. At the same time, a separate cryptographic key provides integrity and optional confidentiality for the media flowing between the Media Distributor and the endpoints.

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

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

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

2. Terminology

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

Terms used throughout this document include:

- Media Distributor: media distribution device that routes media from one endpoint to other endpoints

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

- hop-by-hop: meaning the link from the endpoint to or from the Media Distributor.

- OHB: Original Header Block is an octet string that contains the original values from the RTP header that might have been changed by a Media Distributor.

3. Cryptographic Context

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

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

- 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.

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

3.1. Key Derivation

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

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

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

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

4. Original Header Block

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

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

The OHB has the following syntax (in ABNF):

```
BYTE = %x00-FF

PT = BYTE
SEQ = 2BYTE
Config = BYTE

OHB = ?PT ?SEQ Config
```

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

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

o P: PT is present

o Q: SEQ is present

o M: Marker bit is present

o B: Value of marker bit

o R: Reserved, MUST be set to 0

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

5. RTP Operations

5.1. Encrypting a Packet

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

1. Form an RTP packet. If there are any header extensions, they MUST use [RFC8285].

2. If the packet is for repair mode data, skip to step 6.

3. Form a synthetic RTP packet with the following contents:

    * Header: The RTP header of the original packet with the following modifications:

    * The X bit is set to zero

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

    * Payload: The RTP payload of the original packet

4. Apply the inner cryptographic algorithm to the synthetic RTP packet from the previous step.

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

6. Apply the outer cryptographic algorithm to the RTP packet. If encrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used when encrypting the RTP packet using the outer cryptographic key.

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

5.2. Relaying a Packet

The Media Distributor has the part of the key for the outer (hop-by-hop), but it does not have the part of the key for the (end-to-end) cryptographic algorithm. The cryptographic algorithm and key used to decrypt a packet and any encrypted RTP header extensions would be the same as those used in the endpoint's outer algorithm and key.

In order to modify a packet, the Media Distributor decrypts the packet, modifies the packet, updates the OHB with any modifications not already present in the OHB, and re-encrypts the packet using the cryptographic using the outer (hop-by-hop) key.

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

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

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

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

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

new Sequence Number. If encrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used.

5.3. Decrypting a Packet

To decrypt a packet, the endpoint first decrypts and verifies using the outer (hop-by-hop) cryptographic key, then uses the OHB to reconstruct the original packet, which it decrypts and verifies with the inner (end-to-end) cryptographic key.

1. Apply the outer cryptographic algorithm to the packet. If the integrity check does not pass, discard the packet. The result of this is referred to as the outer SRTP packet. If decrypting RTP header extensions hop-by-hop, then [RFC6904] MUST be used when decrypting the RTP packet using the outer cryptographic key.

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

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

4. Form a new synthetic SRTP packet with:

   * Header = Received header, with the following modifications:

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

   * The X bit is set to zero

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

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

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

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

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

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

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

6. RTCP Operations

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

7. Use with Other RTP Mechanisms

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

7.1. RTX

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

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

7.2. RED

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

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

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

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

7.3. FEC

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

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

7.4. DTMF

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

8. Recommended Inner and Outer Cryptographic Algorithms

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

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

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

The AES-GCM cryptographic algorithm introduces an additional 16 octets to the length of the packet. When using AES-GCM for both the inner and outer cryptographic algorithms, the total additional length is 32 octets. If no other header extensions are present in the packet and the OHB is introduced, that will consume an additional 8 octets. If other extensions are already present, the OHB will consume up to 4 additional octets. For packets in repair mode, the data they are caring is often already encrypted further increasing the size.

9. Security Considerations

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

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

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

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

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

It is obviously critical that the intermediary has only the outer (hop-by-hop) algorithm key and not the half of the key for the the inner (end-to-end) algorithm. We rely on an external key management protocol to assure this property.

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

The security properties for both the inner (end-to-end) and outer (hop-by-hop) key holders are the same as the security properties of classic SRTP.

10. IANA Considerations

10.1. DTLS-SRTP

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

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

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

The SRTP transform parameters for each of these protection are:

```
DOUBLE_AEAD_AES_128_GCM_AEAD_AES_128_GCM
    cipher:                 AES_128_GCM then AES_128_GCM
    cipher_key_length:      256 bits
    cipher_salt_length:     192 bits
    aead_auth_tag_length:   32 octets
    auth_function:          NULL
    auth_key_length:        N/A
    auth_tag_length:        N/A
    maximum lifetime:       at most 2^31 SRTCP packets and
                            at most 2^48 SRTP packets

DOUBLE_AEAD_AES_256_GCM_AEAD_AES_256_GCM
    cipher:                 AES_256_GCM then AES_256_GCM
    cipher_key_length:      512 bits
    cipher_salt_length:     192 bits
    aead_auth_tag_length:   32 octets
    auth_function:          NULL
    auth_key_length:        N/A
    auth_tag_length:        N/A
    maximum lifetime:       at most 2^31 SRTCP packets and
                            at most 2^48 SRTP packets
```

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

11. Acknowledgments

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

12. References

12.1. Normative References

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

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <https://www.rfc-editor.org/info/rfc3711>.

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

[RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <https://www.rfc-editor.org/info/rfc6904>.

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

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

12.2. Informative References

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

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

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

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

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

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

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

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

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

Appendix A. Encryption Overview

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

```
     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