Audio/Video Transport Core Maintenance
Internet-Draft
Intended status: Standards Track
Expires: 13 July 2026

P. Thatcher
Microsoft
Y. Fablet
A. Rosenberg
Apple

9 January 2026

RTP Payload Format for SFrame
draft-ietf-avtcore-rtp-sframe-02

Abstract

This document describes the RTP payload format of SFrame, its use in SDP, and per stream key derivation.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-wg-avtcore.github.io/draft-ietf-avtcore-rtp-sframe/draft-ietf-avtcore-rtp-sframe.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-avtcore-rtp-sframe/.

Discussion of this document takes place on the Audio/Video Transport Core Maintenance Working Group mailing list (mailto:avt@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/avt/. Subscribe at https://www.ietf.org/mailman/listinfo/avt/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-avtcore/draft-ietf-avtcore-rtp-sframe.

Status of This Memo

This Internet-Draft will expire on 13 July 2026.

Table of Contents

1. Introduction

SFrame [RFC9605] describes an end-to-end encryption and authentication mechanism for media data in a multiparty conference call, in which central media servers (SFUs) can access the media metadata needed to make forwarding decisions without having access to the actual media.

This document describes how to packetize a media frame encrypted using SFrame into RTP packets, how to signal support in SDP, and how to derive per SSRC SFrame keys from a common key.

2. Terminology and Notation

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.

3. SFrame format

An SFrame ciphertext comprises a header and encrypted data. The SFrame header has a size varying between 1 to 17 bytes. The encrypted data can be of arbitrary length and is larger than the unencrypted data by a fixed overhead that depends on the encryption algorithm. The overhead can be up to 16 bytes.

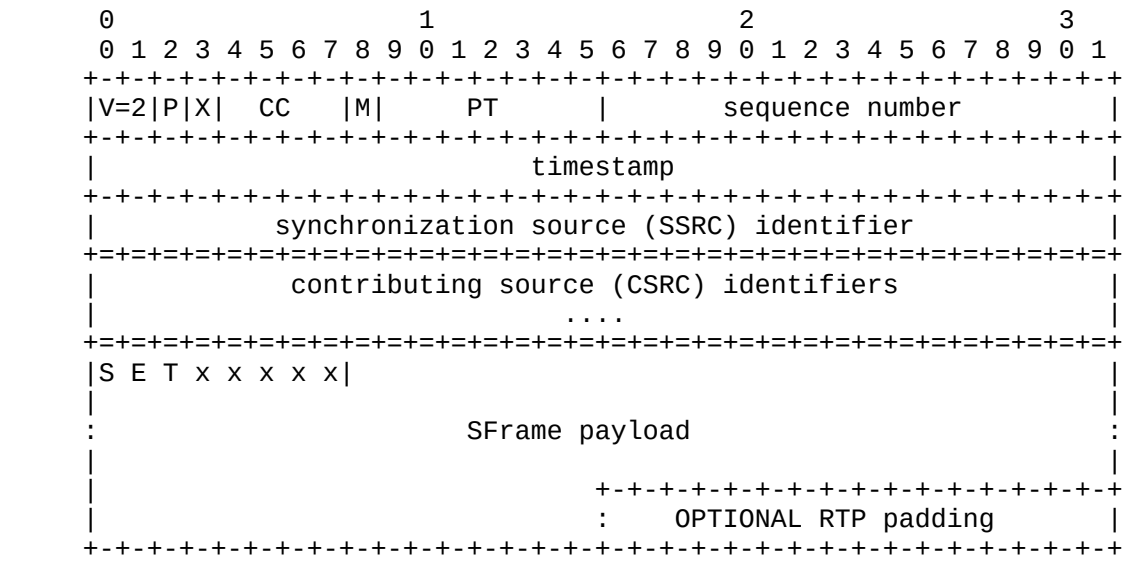
An SFrame ciphertext having an arbitrary long length, an application may decide to partition the data encrypted with SFrame into pieces small enough that the SFrame ciphertext fits in a single RTP packet. We call this per-packet SFrame. This has the advantage of allowing to decrypt the content as soon as received.

An alternative is to encrypt the data, a media frame typically, and send the SFrame ciphertext over several RTP packets. We call this per-frame SFrame. This has the advantage of limiting the SFrame overhead, especially for video frames. This alternative is also compatible with [WebRTC_Encoded_Transform], which is important for backward compatibility of existing services.

The RTP format presented in this document supports both alternatives.

4. RTP Header Usage

The general RTP payload format for SFrame is depicted below.



The first byte of the RTP payload is the SFrame payload descriptor.

The S bit of the SFrame payload descriptor MUST be 0 for all fragments except for the first one of the SFrame frame.

The E bit of the SFrame payload descriptor MUST be 0 for all fragments except for the last one of the SFrame frame.

A SFrame frame is protecting media content that can be either in a packetized format or in a raw format as defined by [WebRTC_Encoded_Transform]. The T bit of the SFrame payload descriptor MUST be 0 if the fragment comes from a raw format as defined by [WebRTC_Encoded_Transform]. The T bit of the SFrame payload descriptor MUST be 1 if the fragment comes from a packetized format. All fragments of a SFrame frame MUST have the same T bit value.

The 5 remaining bits of the SFrame payload descriptor are reserved for future use.

The payload type (PT) identifies the format of the media encrypted with SFrame.

The SSRC, timestamp, marker bit, and CSRCs of the SFrame RTP packets MUST be the same as those of the output of the media-format-specific packetization. The header extensions of the SFrame RTP packets SHOULD be the same as those of the output of the media-format-specific packetization, but some may be omitted if it is known that the omitted header extensions do not need to be duplicated on each SFrame RTP packet.

5. RTP Processing of SFrame frames

5.1. RTP Packetization of SFrame

For a given SFrame frame to be sent, the following steps are done:

1. Let content be the SFrame frame payload to be sent.

2. Let metadata be the SFrame frame metadata. It contains information such as payload type, timestamp, synchronization source, contributing sources, marker bit, header extensions, and payload origin.

3. Fragment the content in a list of payloads so that RTP packets generated from them do not exceed the network maximum transmission unit size.

4. For each payload in the list, run the following steps:

    1. Let descriptor be a SFrame payload descriptor, initialized to a zero byte.

    2. Set the first bit S of the descriptor to 1 if the payload is first in the list and 0 otherwise.

    3. Set the second bit E of the descriptor to 1 if the payload is last in the list and 0 otherwise.

    4. Set the third bit T of the descriptor to 1 if the SFrame frame payload origin is packetized, and 0 if it is raw.

    5. Prepend the payload with the descriptor.

    6. Let packet be a RTP packet with payload.

    7. Set the packet’s RTP header, including header extensions, using the metadata.

8. In particular, if the SFrame frame marker bit is set, set the RTP header marker bit to the value of the second bit E of the descriptor.

9. Send the packet.

SFrame packets can be generated either from RTP media packets or from media frames as defined by [WebRTC_Encoded_Transform]. The two kind of processing are presented below.

5.1.1. Per-frame SFrame sending

For per-frame SFrame, the following steps are done on sender side, with a media frame as input:

1. Generate a list of RTP packets from the media frame using the media-format-specific packetizer.

2. Let packet be the last RTP packet in the list.

3. Let payload be a SFrame ciphertext generated from the media frame data.

4. Let metadata be a SFrame frame metadata.

5. Set the SFrame metadata payload origin to raw.

6. Set the SFrame metadata marker bit if the packet has its marker bit set.

7. Set the SFrame metadata other fields (timestamp, payload type, RTP header extensions...) using the packet RTP header.

8. Send the SFrame frame generated from payload and metadata to the SFrame packetizer.

5.1.2. Per-packet SFrame sending

For per-packet SFrame, the following steps are done on sender side, with a media frame as input:

1. Generate a list of RTP packets from the media frame using the media-format-specific packetizer. The packetizer can be made aware of the SFrame overhead so that packet payloads do not get further split when doing SFrame packetization, to get a 1-1 match between the generated RTP packets and the SFrame encrypted sent RTP packets.

2. For each RTP packet in the list, run the following steps:

    1. Let payload be a SFrame ciphertext generated from the RTP packet payload.

    2. Let metadata be a SFrame frame metadata.

    3. Set the SFrame metadata content origin to packetized.

    4. Set the SFrame metadata other fields using the packet RTP header, including the marker bit.

    5. Send the SFrame frame generated from payload and metadata to the SFrame packetizer.

5.2. RTP depacketization of SFrame

When receiving a RTP SFrame packet, the following steps are done:

1. Add the packet to the set of received RTP packets.

2. Sort the set in order of the RTP sequence number.

3. Search for the smallest list of RTP packets in the set such that:

    1. The packets are consecutive in sequence number.

    2. The first packet of the list has its S bit set to 1.

    3. The last packet of the list has its E bit set to 1.

4. If there is no such list, abort these steps.

5. Remove the selected RTP packets from the set of received RTP packets.

6. If the T bits of each RTP packet of the list do not have the same value, abort these steps.

7. If the payload types of each RTP packet of the list do not have the same value, abort these steps.

8. For each RTP packet of the list, remove the first byte of the packet’s payload.

9. Let the SFrame payload be the concatenation of the payloads of each RTP packet of the list.

10. Let the SFrame metadata be computed from the RTP headers of the RTP packets of the list, in particular:

    1. The metadata’s marker bit is set if the marker bit of the last RTP packet of the list is set.

    2. The metadata’s payload origin is packetized if the T bit value is 1 and raw otherwise.

11. Decrypt the SFrame payload to obtain the media decrypted data.

12. If the frame metadata content origin is raw, aka the T bits of each RTP packet of the list are 0, send the media frame created from the media decrypted data and SFrame metadata to the receiving pipeline.

13. Otherwise, run the following sub-steps:

    1. Let packetizer be a media-format-specific packetizer selected from the frame payload type.

    2. Send the media decrypted data and SFrame metadata to the packetizer.

    3. If the packetizer generates a media frame, send the media frame to the receiving pipeline.

6. SFrame SDP negotiation

SFrame packetization is indicated via a new "a=sframe" SDP attribute defined in this specification. This attribute is used at media level, it does not appear at session level.

The presence of the "a=sframe" attribute in a media section (in either an offer or an answer) indicates that the endpoint is expecting to receive RTP packets encrypted with SFrame for that media section, as defined below.

Once each peer has verified that the other party expects to receive SFrame RTP packets, senders are expected to send SFrame encrypted RTP packets. If one peer expects to use SFrame for a media section and identifies that the other peer does not support it, the peer is expected to stop the transceiver associated with the media section, which will generate a zero port for that m-section.

When SFrame is in use for that media section, it will apply to the relevant media encodings defined for that media section. This includes RTP payload types bound to media packetizers and media

depacketizers as defined in [RFC7656], typically audio formats such as Opus and RTP video formats such as H264. This notably includes RTP payload types representing [WebRTC_Encoded_Transform] encoded video frame formats (https://w3c.github.io/webrtc-encoded-transform/#dom-rtcencodedvideoframe-data) and encoded audio frame formats (https://w3c.github.io/webrtc-encoded-transform/#dom-rtcencodedaudioframe-data).

This does not include RTP-Based Redundancy mechanisms as defined in [RFC7656]. For instance, RTX defined in [RFC4588] will retransmit SFrame-based packets. Forward error correction formats as defined in [RFC5109] will protect the encrypted content. For Redundant Audio Data, known as RED, as defined in [RFC2198], a RED packetizer will take as input SFrame encrypted media data instead of unencrypted media data.

If BUNDLE is in use and the "a=sframe" attribute is present for a media section but not for another media section of the same BUNDLE, payload types for media encodings that are relevant for SFrame MUST NOT be reused between the two media sections.

Questions:

1. Should we precise how RTX/FEC works with SFrame packetization? No impact AFAIK since RTX/FEC would work on packets (whether SFramed or not).

2. Is RED current proposal (transmit SFrame ciphertext blocks) good enough? An alternative is to have SFrame being applied on the entire RED packet payload.

3. Should we allow a=sframe at session level to mean that all media sections want sframe?

Here is an example of SFrame being negotiated for audio (opus and CN) and for video (H264 and VP8):

```
m=audio 50000 RTP/SAVPF 10 11
a=sframe
a=rtpmap:10 opus/48000/2
a=rtpmap:11 CN/8000

m=video 50002 RTP/SAVPF 100 101
a=sframe
a=rtpmap:100 H264/90000
a=rtpmap:101 VP8/90000
```

7. SFrame SDP RTP Key Management

When SFrame is used with SDP and specifies RTP as the media transport, an additional key derivation step MAY be applied to produce a unique key per SDP m= line using the SSRC. The resulting per SSRC stream key is used as the initial key in the session for that m= line and becomes the input to the SFrame derive_key_salt function.

This initial derivation step starts with the base_key of the session. Then, each SSRC stream involved in the SDP session, MUST perform this derivation step to produce the initial SFrame ssrc_key for that stream in the SDP session. This step is performed using HMAC-based Key Derivation Function (HKDF) [RFC5869] as follows:

```
ssrc_key = HKDF-Expand(HKDF-Extract(SSRC, base_key), "SFrame 1.0 RTP Stream", CipherSuite
.Nh)
```

In the derivation of ssrc_key:

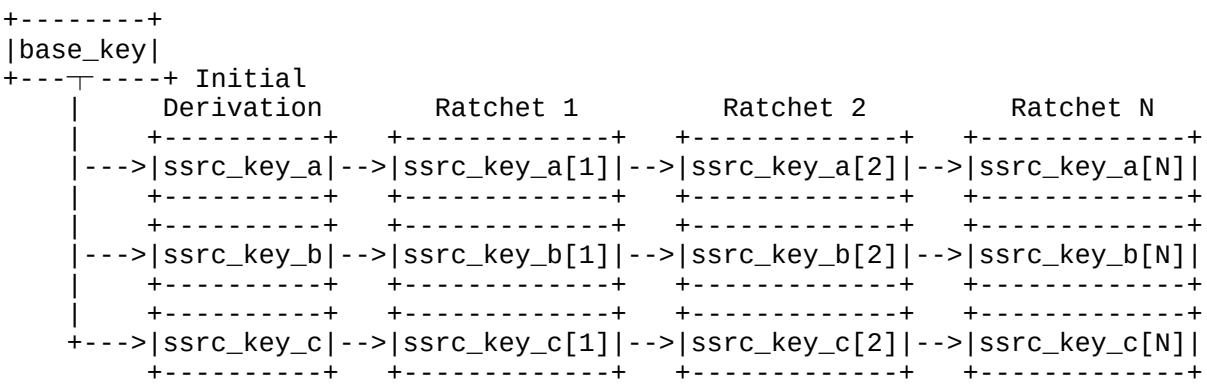
* The SSRC is encoded as a 4-byte big-endian byte sequence and passed as the salt value to HKDF-Extract.

* The same CipherSuite is used for this step as for the per packet / frame step the resulting key will be used with.

8. SFrame Ratcheting with per SSRC keys

If ratcheting is used with this per SSRC key derivation algorithm, the per SSRC key derivation step MUST be done once at the start of the session, and then each subsequent ratchet is done on the per SSRC specific keys using the SFrame ratcheting algorithm described in section 5.1 of [RFC9605]. In this per SSRC key derivation algorithm, the same algorithm defined in section 5.1 of [RFC9605] is used to produce the KID from the key_generation and ratchet_steps inputs. This results in all streams using the same KID, but different keys and CTR values.

This results in a key tree that looks like the following for an offer that has three different SSRC’s a, b, and c.



9. Security Considerations

This document is subject to the security considerations of SFrame.

10. IANA Considerations

This document updates the "attribute-name (formerly "att-field")" subregistry of the "Session Description Protocol (SDP) Parameters" registry (see Section 8.2.4 of [RFC8866]). Specifically, it adds the SDP "a=sframe" attribute for use at the media level.

Contact name: IETF AVT Working Group

Contact email address: avt@ietf.org

Attribute name: sframe

Attribute syntax: This attribute takes no values

Attribute semantics: N/A

Attribute value: N/A

Usage Level: media

Charset dependent: No

Purpose: The presence of this attribute in the SDP indicates that the endpoint is capable of processing SFrame packets

O/A procedures: SDP O/A procedures are defined in section 6

Mux Category: NORMAL

11. 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/rfc/rfc2119>.

[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/rfc/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/rfc/rfc4588>.

[RFC5109] Li, A., Ed., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, DOI 10.17487/RFC5109, December 2007, <https://www.rfc-editor.org/rfc/rfc5109>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC7656] Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and B. Burman, Ed., "A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", RFC 7656, DOI 10.17487/RFC7656, November 2015, <https://www.rfc-editor.org/rfc/rfc7656>.

[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/rfc/rfc8174>.

[RFC8866] Begen, A., Kyzivat, P., Perkins, C., and M. Handley, "SDP: Session Description Protocol", RFC 8866, DOI 10.17487/RFC8866, January 2021, <https://www.rfc-editor.org/rfc/rfc8866>.

[RFC9605] Omara, E., Uberti, J., Murillo, S. G., Barnes, R., Ed., and Y. Fablet, "Secure Frame (SFrame): Lightweight Authenticated Encryption for Real-Time Media", RFC 9605, DOI 10.17487/RFC9605, August 2024, <https://www.rfc-editor.org/rfc/rfc9605>.

[WebRTC_Encoded_Transform]
World Wide Web Consortium, "WebRTC Encoded Transform", May 2025, <https://w3c.github.io/webrtc-encoded-transform/>.

Authors' Addresses

Peter Thatcher
Microsoft
Email: pthatcher@microsoft.com

Youenn Fablet
Apple
Email: youenn@apple.com

Aron Rosenberg
Apple
Email: aron.rosenberg@apple.com