Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 20, 2020

E. Omara
J. Uberti
Google
A. GOUAILLARD
S. Murillo
CoSMo Software

May 19, 2020

Secure Frame (SFrame)
draft-omara-sframe-00

Abstract

This document describes the Secure Frame (SFrame) end-to-end encryption and authentication mechanism for media frames 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. The proposed mechanism differs from other approaches through its use of media frames as the encryptable unit, instead of individual RTP packets, which makes it more bandwidth efficient and also allows use with non-RTP transports.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 20, 2020.

Table of Contents

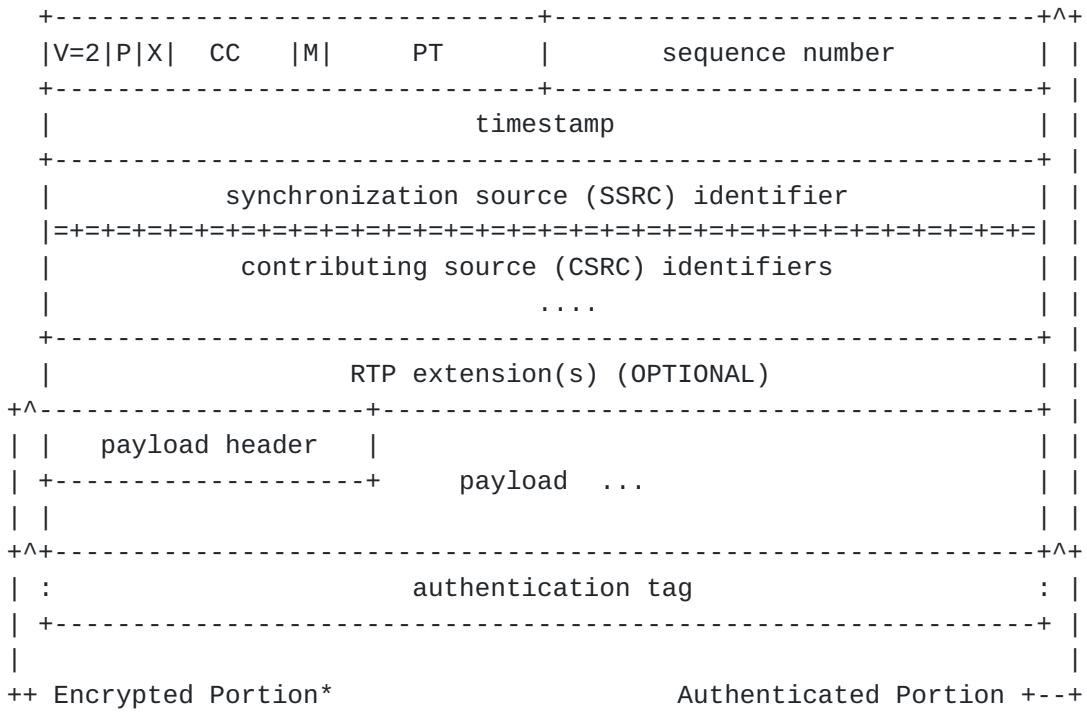
1. Introduction

Modern multi-party video call systems use Selective Forwarding Unit (SFU) servers to efficiently route RTP streams to call endpoints based on factors such as available bandwidth, desired video size, codec support, and other factors. In order for the SFU to work properly though, it needs to be able to access RTP metadata and RTCP feedback messages, which is not possible if all RTP/RTCP traffic is end-to-end encrypted.

As such, two layers of encryptions and authentication are required: 1- Hop-by-hop (HBH) encryption of media, metadata, and feedback messages between the the endpoints and SFU 2- End-to-end (E2E) encryption of media between the endpoints

While DTLS-SRTP can be used as an efficient HBH mechanism, it is inherently point-to-point and therefore not suitable for a SFU context. In addition, given the various scenarios in which video calling occurs, minimizing the bandwidth overhead of end-to-end encryption is also an important goal.

This document proposes a new end-to-end encryption mechanism known as SFrame, specifically designed to work in group conference calls with SFUs.



SRTP packet format

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.

SFU: Selective Forwarding Unit (AKA RTP Switch)

IV: Initialization Vector

MAC: Message Authentication Code

E2EE: End to End Encryption

HBH: Hop By Hop

KMS: Key Management System

3. Goals

SFrame is designed to be a suitable E2EE protection scheme for conference call media in a broad range of scenarios, as outlined by the following goals:

1. Provide an secure E2EE mechanism for audio and video in conference calls that can be used with arbitrary SFU servers.

2. Decouple media encryption from key management to allow SFrame to be used with an arbitrary KMS.

3. Minimize packet expansion to allow successful conferencing in as many network conditions as possible.

4. Independence from the underlying transport, including use in non-RTP transports, e.g., WebTransport.

5. When used with RTP and its associated error resilience mechanisms, i.e., RTX and FEC, require no special handling for RTX and FEC packets.

6. Minimize the changes needed in SFU servers.

7. Minimize the changes needed in endpoints.

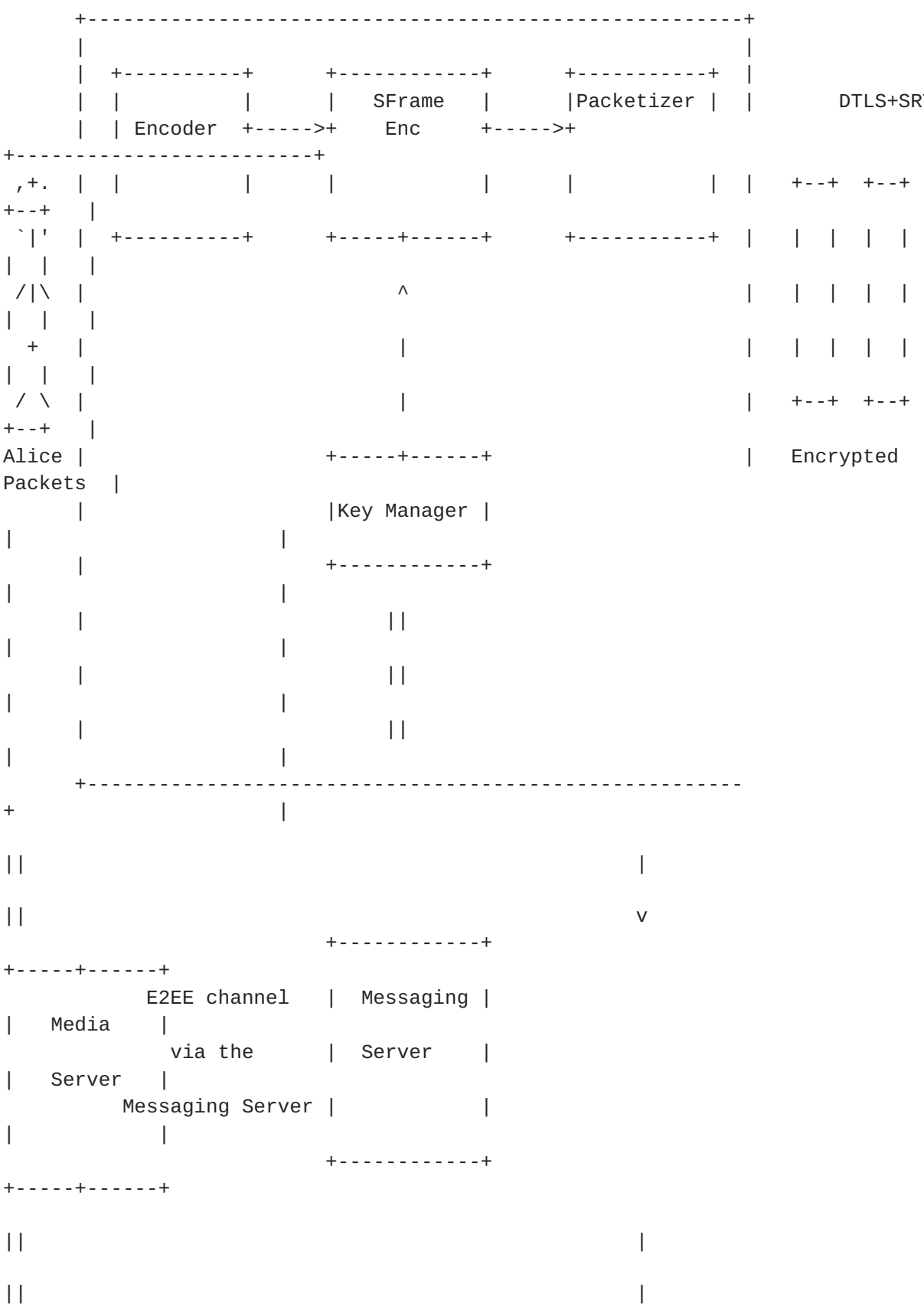
8. Work with the most popular audio and video codecs used in conferencing scenarios.

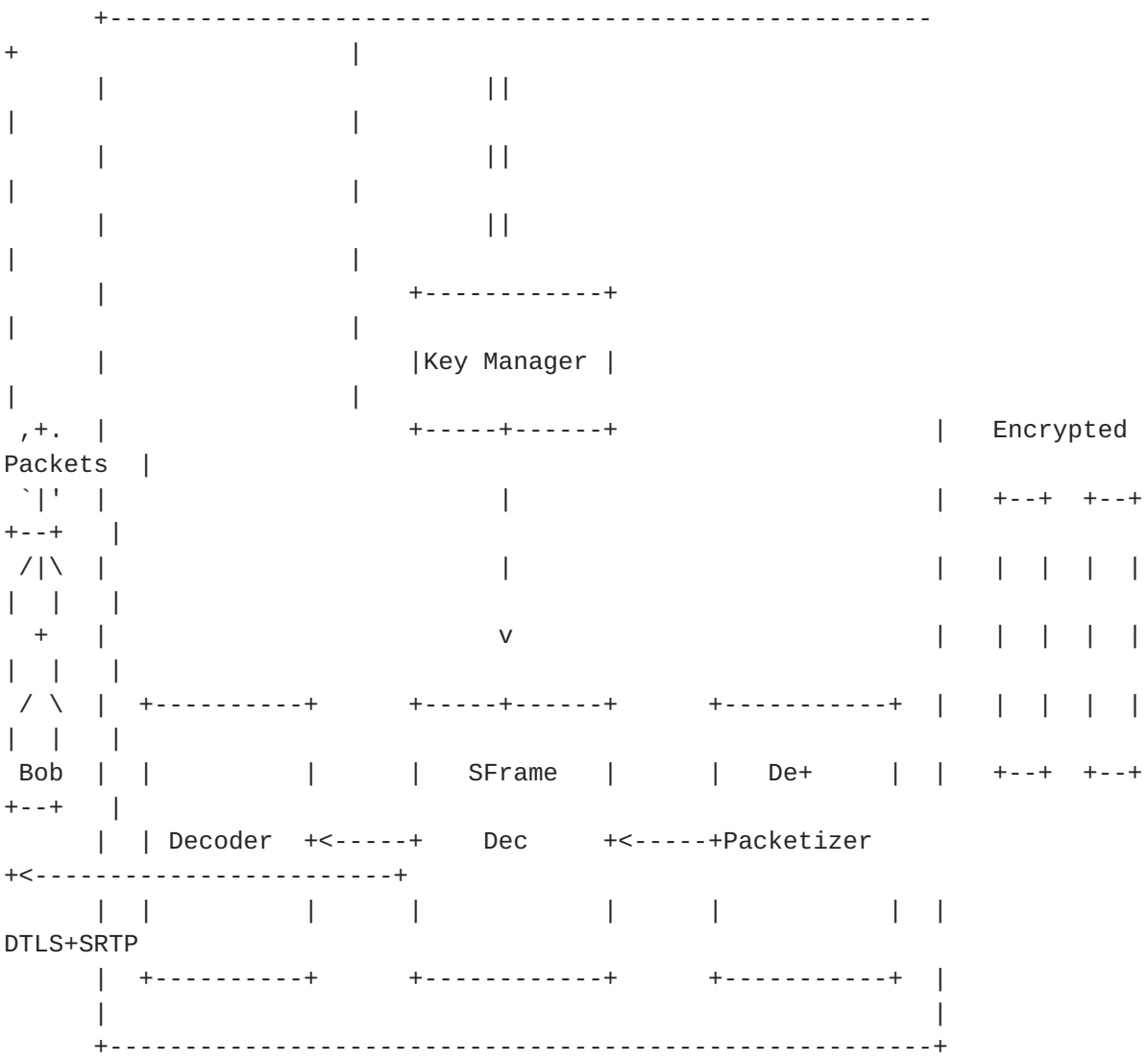
4. SFrame

We propose a frame level encryption mechanism that provides effective end-to-end encryption, is simple to implement, has no dependencies on RTP, and minimizes encryption bandwidth overhead. Because SFrame encrypts the full frame, rather than individual packets, bandwidth overhead is reduced by having a single IV and authentication tag for each media frame.

Also, because media is encrypted prior to packetization, the encrypted frame is packetized using a generic RTP packetizer instead of codec-dependent packetization mechanisms. With this move to a generic packetizer, media metadata is moved from codec-specific mechanisms to a generic frame RTP header extension which, while visible to the SFU, is authenticated end-to-end. This extension includes metadata needed for SFU routing such as resolution, frame beginning and end markers, etc.

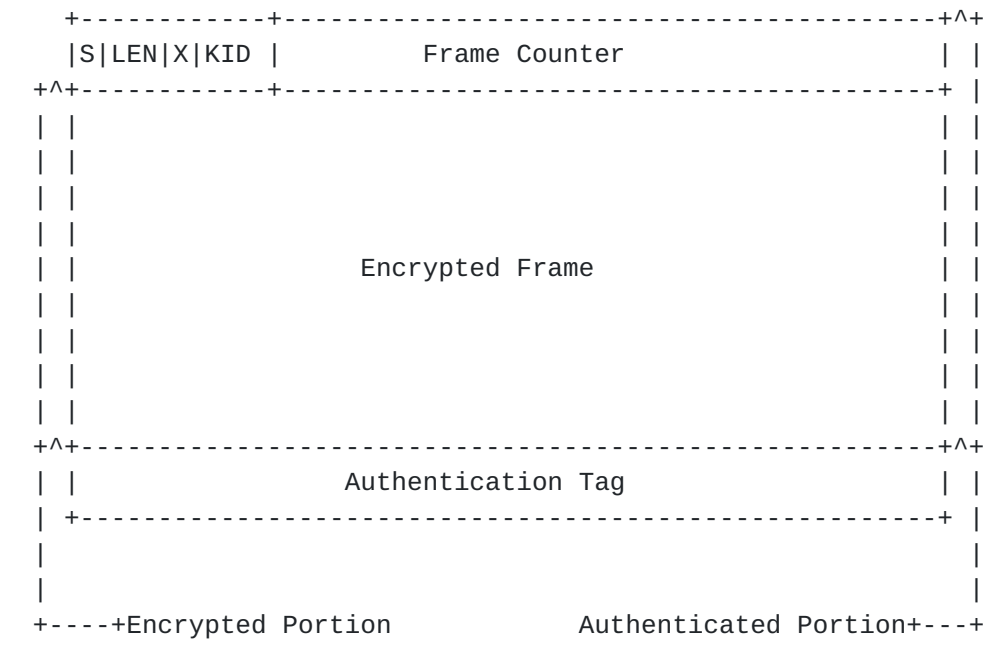
The generic packetizer splits the E2E encrypted media frame into one or more RTP packets and adds the SFrame header to the beginning of the first packet and an auth tag to the end of the last packet.


Encoder
SFrame Enc
Packetizer
DTLS+SRTP
Alice
Encrypted Packets
Key Manager
E2EE channel via the Messaging Server
Messaging Server
Media Server



The E2EE keys used to encrypt the frame are exchanged out of band using a secure E2EE channel.

4.1. SFrame Format

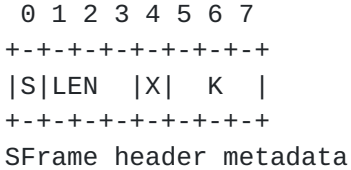


4.2. SFrame Header

Since each endpoint can send multiple media layers, each frame will have a unique frame counter that will be used to derive the encryption IV. The frame counter must be unique and monotonically increasing to avoid IV reuse.

As each sender will use their own key for encryption, so the SFrame header will include the key id to allow the receiver to identify the key that needs to be used for decrypting.

Both the frame counter and the key id are encoded in a variable length format to decrease the overhead, so the first byte in the Sframe header is fixed and contains the header metadata with the following format:

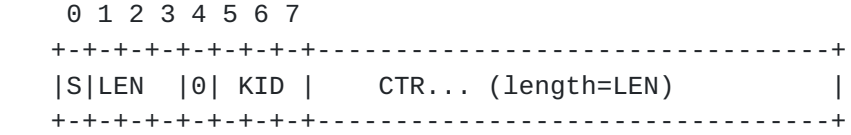


SFrame header metadata

Signature flag (S): 1 bit This field indicates the payload contains a signature if set. Counter Length (LEN): 3 bits This field indicates the length of the CTR fields in bytes. Extended Key Id Flag (X): 1

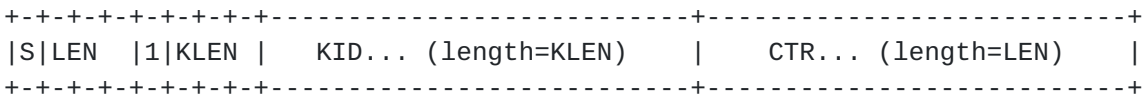
bit Indicates if the key field contains the key id or the key length. Key or Key Length: 3 bits This field contains the key id (KID) if the X flag is set to 0, or the key length (KLEN) if set to 1.

If X flag is 0 then the KID is in the range of 0-7 and the frame counter (CTR) is found in the next LEN bytes:



Key id (KID): 3 bits The key id (0-7). Frame counter (CTR): (Variable length) Frame counter value up to 8 bytes long.

if X flag is 1 then KLEN is the length of the key (KID), that is found after the SFrame header metadata byte. After the key id (KID), the frame counter (CTR) will be found in the next LEN bytes:



Key length (KLEN): 3 bits The key length in bytes. Key id (KID): (Variable length) The key id value up to 8 bytes long. Frame counter (CTR): (Variable length) Frame counter value up to 8 bytes long.

4.3. Encryption Schema

4.3.1. Key Derivation

Each client creates a 32 bytes secret key K and share it with with other participants via an E2EE channel. From K, we derive 3 secrets:

1- Salt key used to calculate the IV

Key = HKDF(K, 'SFrameSaltKey', 16)

2- Encryption key to encrypt the media frame

Key = HKDF(K, 'SFrameEncryptionKey', 16)

3- Authentication key to authenticate the encrypted frame and the media metadata

Key = HKDF(K, 'SFrameAuthenticationKey', 32)

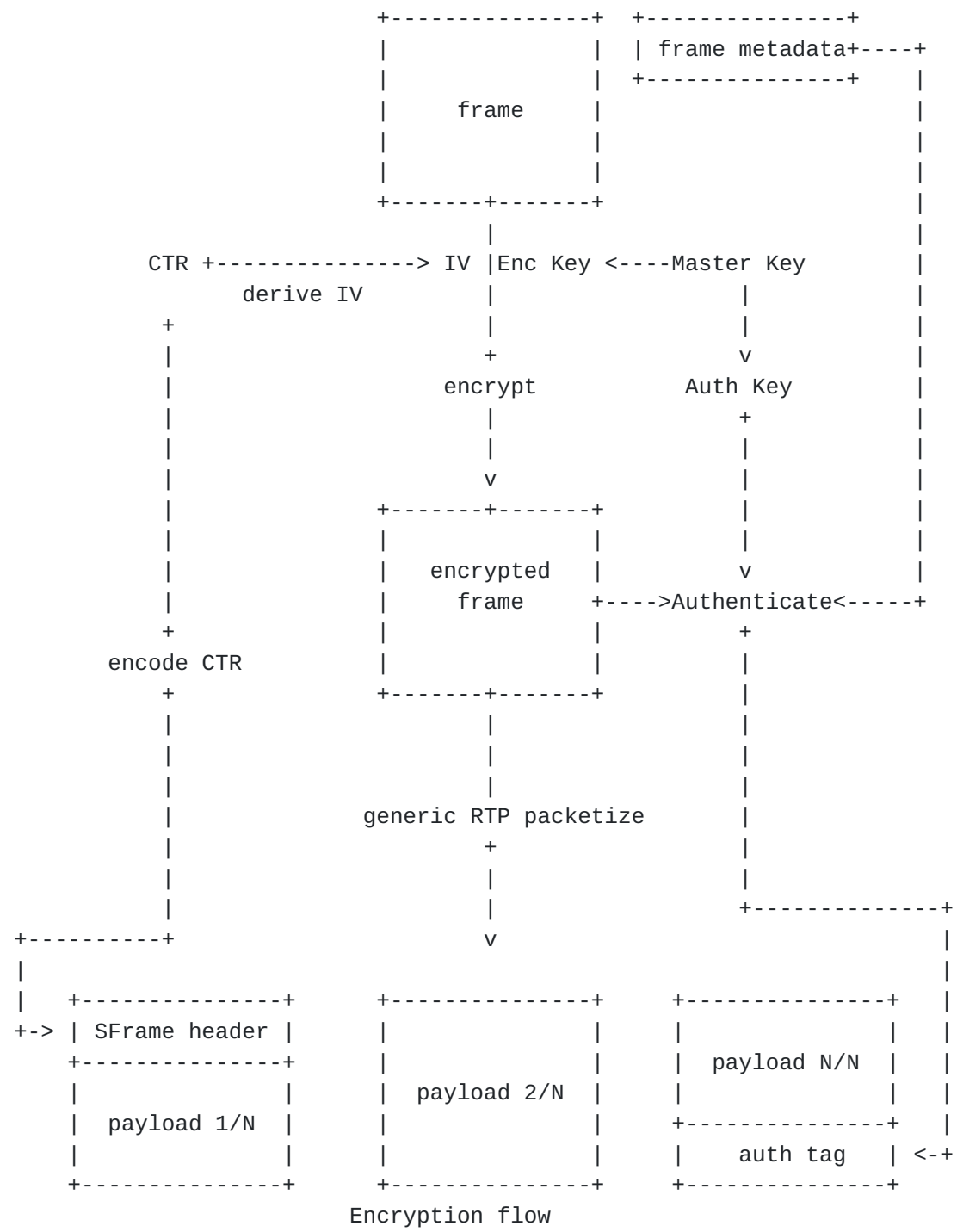
The IV is 128 bits long and calculated from the CTR field of the Frame header:

IV = CTR XOR Salt key

4.3.2. Encryption

After encoding the frame and before packetizing it, the necessary media metadata will be moved out of the encoded frame buffer, to be used later in the RTP generic frame header extension. The encoded frame, the metadata buffer and the frame counter are passed to SFrame encryptor. The encryptor constructs SFrame header using frame counter and key id and derive the encryption IV. The frame is encrypted using the encryption key and the header, encrypted frame, the media metadata and the header are authenticated using the authentication key. The authentication tag is then truncated (If supported by the cipher suite) and prepended at the end of the ciphertext.

The encrypted payload is then passed to a generic RTP packetized to construct the RTP packets and encrypts it using SRTP keys for the HBH encryption to the media server.



Encryption flow

4.3.3. Decryption

The receiving clients buffer all packets that belongs to the same frame using the frame beginning and ending marks in the generic RTP frame header extension, and once all packets are available, it passes it to Frame for decryption. SFrame maintains multiple decryptor objects, one for each client in the call. Initially the client might

not have the mapping between the incoming streams the user's keys, in this case SFrame tries all unmapped keys until it finds one that passes the authentication verification and use it to decrypt the frame. If the client has the mapping ready, it can push it down to SFrame later.

The KeyId field in the SFrame header is used to find the right key for that user, which is incremented by the sender when they switch to a new key.

For frames that are failed to decrypt because there is not key available yet, SFrame will buffer them and retries to decrypt them once a key is received.

4.3.4. Duplicate Frames

Unlike messaging application, in video calls, receiving a duplicate frame doesn't necessary mean the client is under a replay attack, there are other reasons that might cause this, for example the sender might just be sending them in case of packet loss. SFrame decryptors use the highest received frame counter to protect against this. It allows only older frame pithing a short interval to support out of order delivery.

4.3.5. Key Rotation

Because the E2EE keys could be rotated during the call when people join and leave, these new keys are exchanged using the same E2EE secure channel used in the initial key negotiation. Sending new fresh keys is an expensive operation, so the key management component might chose to send new keys only when other clients leave the call and use hash ratcheting for the join case, so no need to send a new key to the clients who are already on the call. SFrame supports both modes

4.3.5.1. Key Ratcheting

When SFrame decryptor fails to decrypt one of the frames, it automatically ratchets the key forward and retries again until one ratchet succeed or it reaches the maximum allowed ratcheting window. If a new ratchet passed the decryption, all previous ratchets are deleted.

K(i) = HKDF(K(i-1), 'SFrameRatchetKey', 32)

4.3.5.2. New Key

SFrame will set the key immediately on the decrypts when it is received and destroys the old key material, so if the key manager sends a new key during the call, it is recommended not to start using it immediately and wait for a short time to make sure it is delivered to all other clients before using it to decrease the number of decryption failure. It is up to the application and the key manager to define how long this period is.

4.4. Authentication

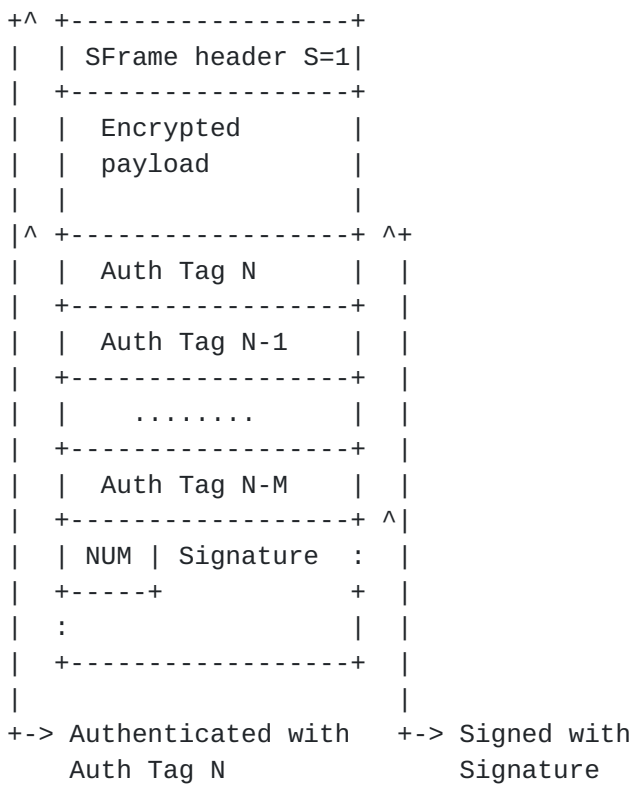
Every client in the call knows the secret key for all other clients so it can decrypt their traffic, it also means a malicious client can impersonate any other client in the call by using the victim key to encrypt their traffic. This might not be a problem for consumer application where the number of clients in the call is small and users know each others, however for enterprise use case where large conference calls are common, an authentication mechanism is needed to protect against malicious users. This authentication will come with extra cost.

Adding a digital signature to each encrypted frame will be an overkill, instead we propose adding signature over multiple frames.

The signature is calculated by concatenating the authentication tags of the frames that the sender wants to authenticate (in reverse sent order) and signing it with the signature key. Signature keys are exchanged out of band along the encryption keys.

```
Signature = Sign(Key, AuthTag(Frame N) || AuthTag(Frame N-1) || ...||
AuthTag(Frame N-M))
```

The authentication tags for the previous frames covered by the signature and the signature itself will be appended at end of the frame, after the current frame authentication tag, in the same order that the signature was calculated, and the SFrame header metadata signature bit (S) will be set to 1.



Encrypted Frame with Signature

Note that the authentication tag for the current frame will only authenticate the SFrame header and the encrypted payload, ant not the signature nor the previous frames's authentication tags (N-1 to N-M) used to calculate the signature.

The last byte (NUM) after the authentication tag list and before the signature indicates the number of the authentication tags from previous frames present in the current frame. All the authentications tags MUST have the same size, which MUST be equal to the authentication tag size of the current frame. The signature is fixed size depending on the signature algorithm used (for example, 64 bytes for Ed25519).

The receiver has to keep track of all the frames received but yet not verified, by storing the authentication tags of each received frame. When a signature is received, the receiver will verify it with the signature key associated to the key id of the frame the signature was sent in. If the verification is successful, the received will mark the frames as authenticated and remove them from the list of the not verified frames. It is up to the application to decide what to do when signature verification fails.

When using SVC, the hash will be calculated over all the frames of the different spatial layers within the same superframe/picture. However the SFU will be able to drop frames within the same stream (either spatial or temporal) to match target bitrate.

If the signature is sent on a frame which layer that is dropped by the SFU, the receiver will not receive it and will not be able to perform the signature of the other received layers.

An easy way of solving the issue would be to perform signature only on the base layer or take into consideration the frame dependency graph and send multiple signatures in parallel (each for a branch of the dependency graph).

In case of simulcast or K-SVC, each spatial layer should be authenticated with different signatures to prevent the SFU to discard frames with the signature info.

In any case, it is possible that the frame with the signature is lost or the SFU drops it, so the receiver MUST be prepared to not receive a signature for a frame and remove it from the pending to be verified list after a timeout.

4.5. Ciphersuites

4.5.1. SFrame

Each SFrame session uses a single ciphersuite that specifies the following primitives:

o A hash function This is used for the Key derivation and frame hashes for signature. We recommend using SHA256 hash function.

o An AEAD encryption algorithm [RFC5116] While any AEAD algorithm can be used to encrypt the frame, we recommend using algorithms with safe MAC truncation like AES-CTR and HMAC to reduce the per-frame overhead. In this case we can use 80 bits MAC for video frames and 32 bits for audio frames similar to DTLS-SRTP cipher suites:

1- AES_CM_128_HMAC_SHA256_80

2- AES_CM_128_HMAC_SHA256_32

o [Optional] A signature algorithm If signature is supported, we recommend using ed25519

4.5.2. DTLS-SRTP

SRTP is used as an HBH encryption, since the media payload is already encrypted, and SRTP only protects the RTP headers, one implementation could use 4 bytes outer auth tag to decrease the overhead, however it is up to the application to use other ciphers like AES-128-GCM with full authentication tag.

5. Key Management

SFrame must be integrated with an E2EE key management framework to exchange and rotate the encryption keys. This framework will maintain a group of participant endpoints who are in the call. At call setup time, each endpoint will create a fresh key material and optionally signing key pair for that call and encrypt the key material and the public signing key to every other endpoints. They encrypted keys are delivered by the messaging delivery server using a reliable channel.

The KMS will monitor the group changes, and exchange new keys when necessary. It is up to the application to define this group, for example one application could have ephemeral group for every call and keep rotating key when end points joins or leave the call, while another application could have a persisted group that can be used for multiple calls and exchange keys with all group endpoints for every call.

When a new key material is created during the call, we recommend not to start using it immediately in SFrame to give time for the new keys to be delivered. If the application supports delivery receipts, it can be used to track if the key is delivered to all other endpoints on the call before using it.

Keys must have a sequential id starting from 0 and incremented eery time a new key is generated for this endpoint. The key id will be added in the SFrame header during encryption, so the recipient know which key to use for the decryption.

5.1. MLS-SFrame

While any other E2EE KMS can be used with SFrame, there is a big advantage if it is used with [MLSARCH] which natively supports very large groups efficiently. When [MLSPROTO] is used, the endpoints keys (AKA Application secret) can be used directly for SFrame without the need to exchange separate key material. The application secret is rotated automatically by [MLSPROTO] when group membership changes.

6. Media Considerations

6.1. SFU

Selective Forwarding Units (SFUs) as described in https://tools.ietf.org/html/rfc7667#section-3.7 receives the RTP streams from each participant and selects which ones should be forwarded to each of the other participants. There are several approaches about how to do this stream selection but in general, in order to do so, the SFU needs to access metadata associated to each frame and modify the RTP information of the incoming packets when they are transmitted to the received participants.

This section describes how this normal SFU modes of operation interacts with the E2EE provided by SFrame

6.1.1. LastN and RTP stream reuse

The SFU may choose to send only a certain number of streams based on the voice activity of the participants. To reduce the number of SDP O/A required to establish a new RTP stream, the SFU may decide to reuse previously existing RTP sessions or even pre-allocate a predefined number of RTP streams and choose in each moment in time which participant media will be sending through it. This means that in the same RTP stream (defined by either SSRC or MID) may carry media from different streams of different participants. As different keys are used by each participant for encoding their media, the receiver will be able to verify which is the sender of the media coming within the RTP stream at any given point if time, preventing the SFU trying to impersonate any of the participants with another participant's media. Note that in order to prevent impersonation by a malicious participant (not the SFU) usage of the signature is required. In case of video, the a new signature should be started each time a key frame is sent to allow the receiver to identify the source faster after a switch.

6.1.2. Simulcast

When using simulcast, the same input image will produce N different encoded frames (one per simulcast layer) which would be processed independently by the frame encryptor and assigned an unique counter for each.

6.1.3. SVC

In both temporal and spatial scalability, the SFU may choose to drop layers in order to match a certain bitrate or forward specific media sizes or frames per second. In order to support it, the sender MUST

encode each spatial layer of a given picture in a different frame. That is, an RTP frame may contain more than one SFrame encrypted frame with an incrementing frame counter.

6.2. Video Key Frames

Forward and Post-Compromise Security requires that the e2ee keys are updated anytime a participant joins/leave the call.

The key exchange happens async and on a different path than the SFU signaling and media. So it may happen that when a new participant joins the call and the SFU side requests a key frame, the sender generates the e2ee encrypted frame with a key not known by the receiver, so it will be discarded. When the sender updates his sending key with the new key, it will send it in a non-key frame, so the receiver will be able to decrypt it, but not decode it.

Receiver will re-request an key frame then, but due to sender and sfu policies, that new key frame could take some time to be generated.

If the sender sends a key frame when the new e2ee key is in use, the time required for the new participant to display the video is minimized.

6.3. Partial Decoding

Some codes support partial decoding, where it can decrypt individual packets without waiting for the full frame to arrive, with SFrame this won't be possible because the decoder will not access the packets until the entire frame is arrived and decrypted.

7. Overhead

The encryption overhead will vary between audio and video streams, because in audio each packet is considered a separate frame, so it will always have extra MAC and IV, however a video frame usually consists of multiple RTP packets. The number of bytes overhead per frame is calculated as the following 1 + FrameCounter length + 4 The constant 1 is the SFrame header byte and 4 bytes for the HBH authentication tag for both audio and video packets.

7.1. Audio

Using three different audio frame durations 20ms (50 packets/s) 40ms (25 packets/s) 100ms (10 packets/s) Up to 3 bytes frame counter (3.8 days of data for 20ms frame duration) and 4 bytes fixed MAC length.

Counter len	Packets	Overhead bps@20ms	Overhead bps@40ms	Overhead bps@100ms
1	0-255	2400	1200	480
2	255 - 65K	2800	1400	560
3	65K - 16M	3200	1600	640

7.2. Video

The per-stream overhead bits per second as calculated for the following video encodings: 30fps@1000Kbps (4 packets per frame) 30fps@512Kbps (2 packets per frame) 15fps@200Kbps (2 packets per frame) 7.5fps@30Kbps (1 packet per frame) Overhead bps = (Counter length + 1 + 4) * 8 * fps

Counter len	Frames	Overhead bps@30fps	Overhead bps@15fps	Overhead bps@7.5fps
1	0-255	1440	1440	720
2	256 - 65K	1680	1680	840
3	56K - 16M	1920	1920	960
4	16M - 4B	2160	2160	1080

7.3. SFrame vs PERC-lite

[PERC] has significant overhead over SFrame because the overhead is per packet, not per frame, and OHB (Original Header Block) which duplicates any RTP header/extension field modified by the SFU. [PERCLITE] <https://mailarchive.ietf.org/arch/msg/perc/SB0qMHWz6EsDtz3yIEX0HWp5IEY/> is slightly better because it doesn't use the OHB anymore, however it still does per packet encryption using SRTP. Below the the overheard in [PERCLITE] implemented by Cosmos Software which uses extra 11 bytes per packet to preserve the PT, SEQ_NUM, TIME_STAMP and SSRC fields in addition to the extra MAC tag per packet.

OverheadPerPacket = 11 + MAC length Overhead bps = PacketPerSecond * OverHeadPerPacket * 8

Similar to SFrame, we will assume the HBH authentication tag length will always be 4 bytes for audio and video even though it is not the case in this [PERCLITE] implementation

7.3.1. Audio

Overhead bps@20ms	Overhead bps@40ms	Overhead bps@100ms
6000	3000	1200

7.3.2. Video

Overhead bps@30fps (4 packets per frame)	Overhead bps@15fps (2 packets per frame)	Overhead bps@7.5fps (1 packet per frame)
14400	7200	3600

For a conference with a single incoming audio stream (@ 50 pps) and 4 incoming video streams (@200 Kbps), the savings in overhead is 34800 - 9600 = ~25 Kbps, or ~3%.

8. Security Considerations

8.1. Key Management

Key exchange mechanism is out of scope of this document, however every client MUST change their keys when new clients joins or leaves the call for "Forward Secrecy" and "Post Compromise Security".

8.2. Authentication tag length

The cipher suites defined in this draft use short authentication tags for encryption, however it can easily support other ciphers with full authentication tag if the short ones are proved insecure.

9. IANA Considerations

This document makes no requests of IANA.

10. References

10.1. 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>.

[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>.

10.2. Informative References

[MLSARCH] Omara, E., Barnes, R., Rescorla, E., Inguva, S., Kwon, A., and A. Duric, "Messaging Layer Security Architecture", 2020.

[MLSPROTO] Barnes, R., Millican, J., Omara, E., Cohn-Gordon, K., and R. Robert, "Messaging Layer Security Protocol", 2020.

[PERC] Jennings, C., Jones, P., Barnes, R., and A. Roach, "PERC", 2020, <https://datatracker.ietf.org/doc/rfc8723/>.

[PERCLITE] GOUAILLARD, A. and S. Murillo, "PERC-Lite", 2020, <https://tools.ietf.org/html/draft-murillo-perc-lite-01>.

Authors' Addresses

Emad Omara
Google

Email: emadomara@google.com

Justin Uberti
Google

Email: juberti@google.com

Alexandre GOUAILLARD
CoSMo Software

Email: Alex.GOUAILLARD@cosmosoftware.io

Sergio Garcia Murillo
CoSMo Software

Email: sergio.garcia.murillo@cosmosoftware.io