Workgroup: Network Working Group
Internet-Draft: draft-ietf-sframe-enc-09
Published: 4 April 2024
Intended Status: Standards Track
Expires: 6 October 2024
Authors: E. Omara (Apple), J. Uberti (Google), S. Murillo (CoSMo Software), R. L. Barnes, Ed. (Cisco), Y. Fablet (Apple)

Secure Frame (SFrame)

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 (selective forwarding units or SFUs) can access the media metadata needed to make forwarding decisions without having access to the actual media.

The proposed mechanism differs from the Secure Real-Time Protocol (SRTP) in that it is independent of RTP (thus compatible with non-RTP media transport) and can be applied to whole media frames in order to be more bandwidth efficient.

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://sframe-wg.github.io/sframe/draft-ietf-sframe-enc.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-sframe-enc/.

Discussion of this document takes place on the Secure Media Frames Working Group mailing list (mailto:sframe@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/sframe/. Subscribe at https://www.ietf.org/mailman/listinfo/sframe/.

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

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 6 October 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction
2. Terminology
3. Goals
4. SFrame
  4.1. Application Context
  4.2. SFrame Ciphertext
  4.3. SFrame Header
  4.4. Encryption Schema
    4.4.1. Key Selection
    4.4.2. Key Derivation
    4.4.3. Encryption
    4.4.4. Decryption
  4.5. Cipher Suites
    4.5.1. AES-CTR with SHA2
5. Key Management
  5.1. Sender Keys
  5.2. MLS
6. Media Considerations
  6.1. Selective Forwarding Units
    6.1.1. LastN and RTP stream reuse
    6.1.2. Simulcast
    6.1.3. SVC

6.2. Video Key Frames
6.3. Partial Decoding
7. Security Considerations
7.1. No Header Confidentiality
7.2. No Per-Sender Authentication
7.3. Key Management
7.4. Replay
7.5. Risks due to Short Tags
8. IANA Considerations
8.1. SFrame Cipher Suites
9. Application Responsibilities
9.1. Header Value Uniqueness
9.2. Key Management Framework
9.3. Anti-Replay
9.4. Metadata
10. References
10.1. Normative References
10.2. Informative References
Appendix A. Acknowledgements
Appendix B. Example API
Appendix C. Overhead Analysis
C.1. Assumptions
C.2. Audio
C.3. Video
C.4. Conferences
C.5. SFrame over RTP
Appendix D. Test Vectors
D.1. Header encoding/decoding
D.2. AEAD encryption/decryption using AES-CTR and HMAC
D.3. SFrame encryption/decryption
Contributors
Authors' Addresses

1. Introduction

Modern multi-party video call systems use Selective Forwarding Unit (SFU) servers to efficiently route media streams to call endpoints based on factors such as available bandwidth, desired video size, codec support, and other factors. An SFU typically does not need access to the media content of the conference, allowing for the media to be "end-to-end" encrypted so that it cannot be decrypted by the SFU. In order for the SFU to work properly, though, it usually 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 encryption and authentication are required:

1. Hop-by-hop (HBH) encryption of media, metadata, and feedback messages between the endpoints and SFU

2. End-to-end (E2E) encryption (E2EE) of media between the endpoints

The Secure Real-Time Protocol (SRTP) is already widely used for HBH encryption [RFC3711]. The SRTP "double encryption" scheme defines a way to do E2E encryption in SRTP [RFC8723]. Unfortunately, this scheme has poor efficiency and high complexity, and its entanglement with RTP makes it unworkable in several realistic SFU scenarios.

This document proposes a new E2EE protection scheme known as SFrame, specifically designed to work in group conference calls with SFUs. SFrame is a general encryption framing that can be used to protect media payloads, agnostic of transport.

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.

MAC: Message Authentication Code

E2EE: End to End Encryption

HBH: Hop By Hop

We use "Selective Forwarding Unit (SFU)" and "media stream" in a less formal sense than in [RFC7656]. An SFU is a selective switching function for media payloads, and a media stream a sequence of media payloads, in both cases regardless of whether those media payloads are transported over RTP or some other protocol.

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 a 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 key management system.

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 [I-D.ietf-webtrans-overview].

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

This document defines an encryption mechanism that provides effective E2EE, is simple to implement, has no dependencies on RTP, and minimizes encryption bandwidth overhead. This section describes how the mechanism works, including details of how applications utilize SFrame for media protection, as well as the actual mechanics of E2EE for protecting media.

4.1. Application Context

SFrame is a general encryption framing, intended to be used as an E2EE layer over an underlying HBH-encrypted transport such as SRTP or QUIC [RFC3711][I-D.ietf-moq-transport].

The scale at which SFrame encryption is applied to media determines the overall amount of overhead that SFrame adds to the media stream, as well as the engineering complexity involved in integrating SFrame into a particular environment. Two patterns are common: Either using SFrame to encrypt whole media frames (per-frame) or individual transport-level media payloads (per-packet).

For example, Figure 1 shows a typical media sender stack that takes media from some source, encodes it into frames, divides those frames into media packets, and then sends those payloads in SRTP packets. The receiver stack performs the reverse operations, reassembling frames from SRTP packets and decoding. Arrows indicate two different ways that SFrame protection could be integrated into this media stack, to encrypt whole frames or individual media packets.

Applying SFrame per-frame in this system offers higher efficiency, but may require a more complex integration in environments where depacketization relies on the content of media packets. Applying SFrame per-packet avoids this complexity, at the cost of higher

bandwidth consumption. Some quantitative discussion of these trade-offs is provided in Appendix C.

As noted above, however, SFrame is a general media encapsulation, and can be applied in other scenarios. The important thing is that the sender and receivers of an SFrame-encrypted object agree on that object's semantics. SFrame does not provide this agreement; it must be arranged by the application.

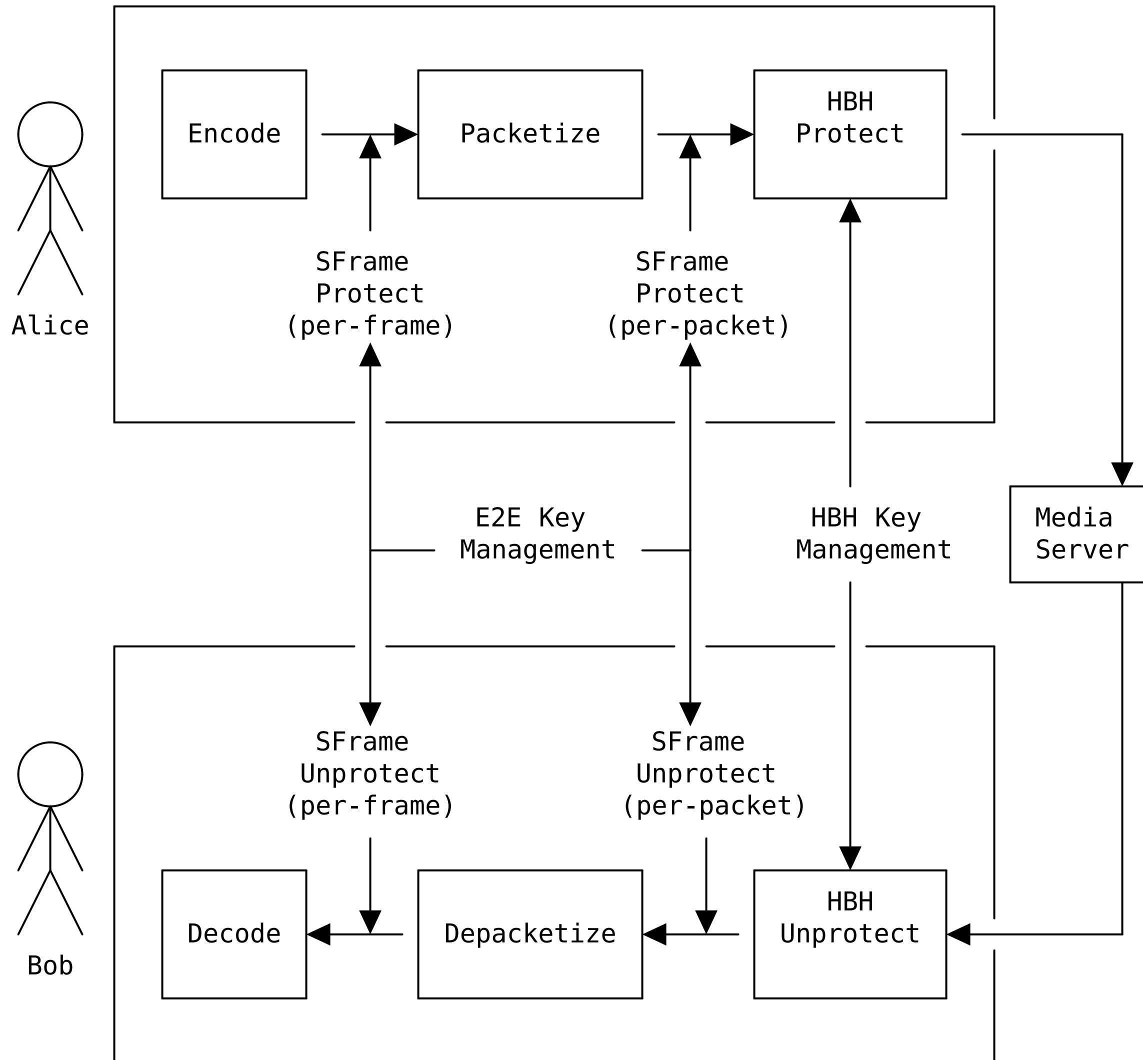


Figure 1

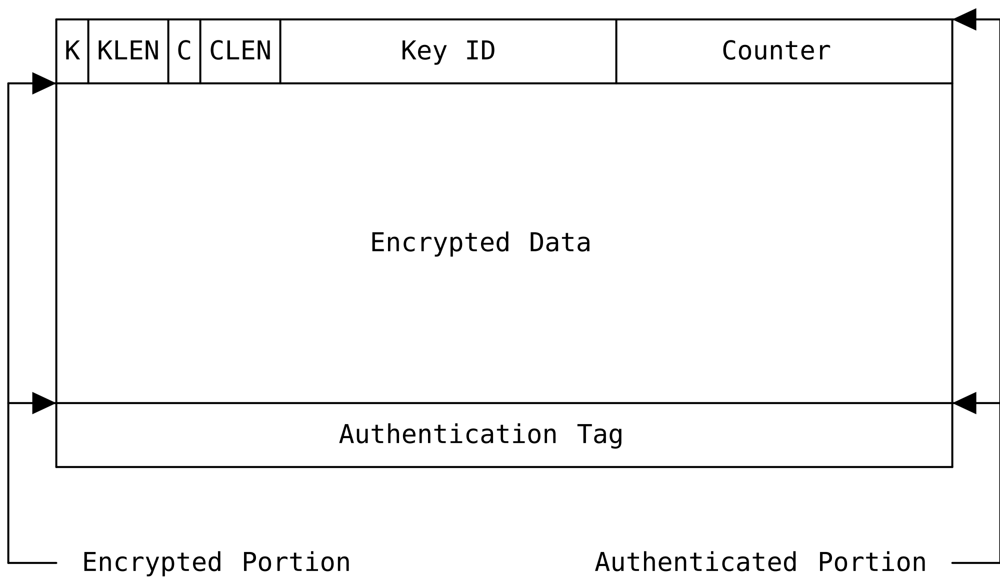
Like SRTP, SFrame does not define how the keys used for SFrame are exchanged by the parties in the conference. Keys for SFrame might be distributed over an existing E2E-secure channel (see Section 5.1), or derived from an E2E-secure shared secret (see Section 5.2). The key management system MUST ensure that each key used for encrypting media

is used by exactly one media sender, in order to avoid reuse of nonces.

4.2. SFrame Ciphertext

An SFrame ciphertext comprises an SFrame header followed by the output of an AEAD encryption of the plaintext [RFC5116], with the header provided as additional authenticated data (AAD).

The SFrame header is a variable-length structure described in detail in Section 4.3. The structure of the encrypted data and authentication tag are determined by the AEAD algorithm in use.



When SFrame is applied per-packet, the payload of each packet will be an SFrame ciphertext. When SFrame is applied per-frame, the SFrame ciphertext representing an encrypted frame will span several packets, with the header appearing in the first packet and the authentication tag in the last packet. It is the responsibility of the application to reassemble an encrypted frame from individual packets, accounting for packet loss and reordering as necessary.

4.3. SFrame Header

The SFrame header specifies two values from which encryption parameters are derived:

* A Key ID (KID) that determines which encryption key should be used

* A counter (CTR) that is used to construct the nonce for the encryption

Applications MUST ensure that each (KID, CTR) combination is used for exactly one SFrame encryption operation. A typical approach to achieving this guarantee is outlined in Section 9.1.

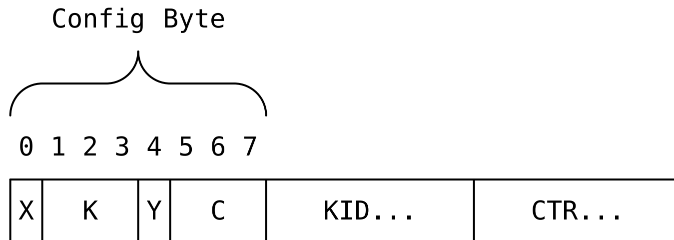


Figure 2: SFrame header

The SFrame Header has the overall structure shown in Figure 2. The first byte is a "config byte", with the following fields:

Extended Key Id Flag (X, 1 bit): Indicates if the K field contains the key id or the Key ID length.

Key or Key Length (K, 3 bits): If the X flag is set to 0, this field contains the Key ID. If the X flag is set to 1, then it contains the length of the Key ID, minus one.

Extended Counter Flag (Y, 1 bit): Indicates if the C field contains the counter or the counter length.

Counter or Counter Length (C, 3 bits): This field contains the counter (CTR) if the Y flag is set to 0, or the counter length, minus one, if set to 1.

The Key ID and Counter fields are encoded as compact unsigned integers in network (big-endian) byte order. If the value of one of these fields is in the range 0-7, then the value is carried in the corresponding bits of the config byte (K or C) and the corresponding flag (X or Y) is set to zero. Otherwise, the value MUST be encoded with the minimum number of bytes required and appended after the configuration byte, with the Key ID first and Counter second. The header field (K or C) is set to the number of bytes in the encoded value, minus one. The value 000 represents a length of 1, 001 a length of 2, etc. This allows a 3-bit length field to represent the value lengths 1-8.

The SFrame header can thus take one of the four forms shown in Figure 3, depending on which of the X and Y flags are set.

```
KID < 8, CTR < 8:
+-+-----+-+-----+
|0| KID |0| CTR |
+-+-----+-+-----+

KID < 8, CTR >= 8:
+-+-----+-+-----+-------------------------+
|0| KID |1|CLEN |  CTR... (length=CLEN)   |
+-+-----+-+-----+-------------------------+

KID >= 8, CTR < 8:
+-+-----+-+-----+-------------------------+
|1|KLEN |0| CTR |  KID... (length=KLEN)   |
+-+-----+-+-----+-------------------------+

KID >= 8, CTR >= 8:
+-+-----+-+-----+-------------------------+-------------------------+
|1|KLEN |1|CLEN |  KID... (length=KLEN)   |  CTR... (length=CLEN)   |
+-+-----+-+-----+-------------------------+-------------------------+
```

Figure 3: Forms of Encoded SFrame Header

4.4. Encryption Schema

SFrame encryption uses an AEAD encryption algorithm and hash function defined by the cipher suite in use (see Section 4.5). We will refer to the following aspects of the AEAD and the hash algorithm below:

* AEAD.Encrypt and AEAD.Decrypt - The encryption and decryption functions for the AEAD. We follow the convention of RFC 5116 [RFC5116] and consider the authentication tag part of the ciphertext produced by AEAD.Encrypt (as opposed to a separate field as in SRTP [RFC3711]).

* AEAD.Nk - The size in bytes of a key for the encryption algorithm

* AEAD.Nn - The size in bytes of a nonce for the encryption algorithm

* AEAD.Nt - The overhead in bytes of the encryption algorithm (typically the size of a "tag" that is added to the plaintext)

* AEAD.Nka - For cipher suites using the compound AEAD described in Section 4.5.1, the size in bytes of a key for the underlying AES-CTR algorithm

* Hash.Nh - The size in bytes of the output of the hash function

4.4.1. Key Selection

Each SFrame encryption or decryption operation is premised on a single secret base_key, which is labeled with an integer KID value signaled in the SFrame header.

The sender and receivers need to agree on which base_key should be used for a given KID. Moreover, senders and receivers need to agree on whether a base_key will be used for encryption or decryption only. The process for provisioning base_key values and their KID values is beyond the scope of this specification, but its security properties will bound the assurances that SFrame provides. For example, if SFrame is used to provide E2E security against intermediary media nodes, then SFrame keys need to be negotiated in a way that does not make them accessible to these intermediaries.

For each known KID value, the client stores the corresponding symmetric key base_key. For keys that can be used for encryption, the client also stores the next counter value CTR to be used when encrypting (initially 0).

When encrypting a plaintext, the application specifies which KID is to be used, and the counter is incremented after successful encryption. When decrypting, the base_key for decryption is selected from the available keys using the KID value in the SFrame Header.

A given base_key MUST NOT be used for encryption by multiple senders. Such reuse would result in multiple encrypted frames being generated with the same (key, nonce) pair, which harms the protections provided by many AEAD algorithms. Implementations MUST mark each base_key as usable for encryption or decryption, never both.

Note that the set of available keys might change over the lifetime of a real-time session. In such cases, the client will need to manage key usage to avoid media loss due to a key being used to encrypt before all receivers are able to use it to decrypt. For example, an application may make decryption-only keys available immediately, but delay the use of keys for encryption until (a) all receivers have acknowledged receipt of the new key or (b) a timeout expires.

4.4.2. Key Derivation

SFrame encryption and decryption use a key and salt derived from the base_key associated to a KID. Given a base_key value, the key and salt are derived using HKDF [RFC5869] as follows:

```
def derive_key_salt(KID, base_key):
  sframe_secret = HKDF-Extract("", base_key)

  sframe_key_label = "SFrame 1.0 Secret key " + KID + cipher_suite
  sframe_key = HKDF-Expand(sframe_secret, sframe_key_label, AEAD.Nk)

  sframe_salt_label = "SFrame 1.0 Secret salt " + KID + cipher_suite
  sframe_salt = HKDF-Expand(sframe_secret, sframe_salt_label, AEAD.Nn)

  return sframe_key, sframe_salt
```

In the derivation of sframe_secret:

* The + operator represents concatenation of byte strings.

* The KID value is encoded as an 8-byte big-endian integer, not the compressed form used in the SFrame header.

* The cipher_suite value is a 2-byte big-endian integer representing the cipher suite in use (see Section 8.1).

The hash function used for HKDF is determined by the cipher suite in use.

4.4.3. Encryption

SFrame encryption uses the AEAD encryption algorithm for the cipher suite in use. The key for the encryption is the sframe_key and the nonce is formed by XORing the sframe_salt with the current counter, encoded as a big-endian integer of length AEAD.Nn.

The encryptor forms an SFrame header using the CTR, and KID values provided. The encoded header is provided as AAD to the AEAD encryption operation, together with application-provided metadata about the encrypted media (see Section 9.4).

```
def encrypt(CTR, KID, metadata, plaintext):
  sframe_key, sframe_salt = key_store[KID]

  # encode_big_endian(x, n) produces an n-byte string encoding the integ
  # big-endian byte order.
  ctr = encode_big_endian(CTR, AEAD.Nn)
  nonce = xor(sframe_salt, CTR)

  # encode_sframe_header produces a byte string encoding the provided KI
  # CTR values into an SFrame Header.
  header = encode_sframe_header(CTR, KID)
  aad = header + metadata

  ciphertext = AEAD.Encrypt(sframe_key, nonce, aad, plaintext)
  return header + ciphertext
```

For example, the metadata input to encryption allows for frame metadata to be authenticated when SFrame is applied per-frame. After encoding the frame and before packetizing it, the necessary media metadata will be moved out of the encoded frame buffer, to be sent in some channel visible to the SFU (e.g., an RTP header extension).

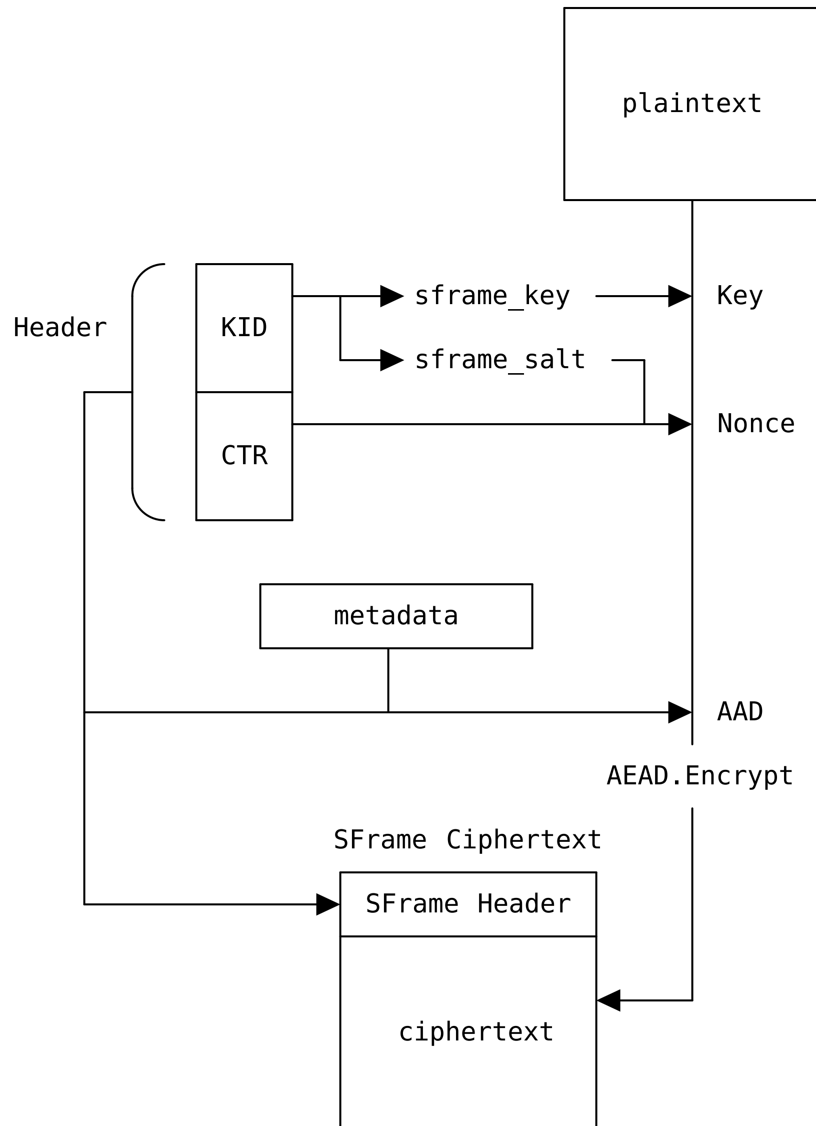


Figure 4: Encrypting an SFrame Ciphertext

4.4.4. Decryption

Before decrypting, a receiver needs to assemble a full SFrame ciphertext. When an SFrame ciphertext may be fragmented into multiple parts for transport (e.g., a whole encrypted frame sent in multiple SRTP packets), the receiving client collects all the fragments of the ciphertext, using appropriate sequencing and start/end markers in the transport. Once all of the required fragments are available, the client reassembles them into the SFrame ciphertext, then passes the ciphertext to SFrame for decryption.

The KID field in the SFrame header is used to find the right key and salt for the encrypted frame, and the CTR field is used to construct the nonce. The SFrame decryption procedure is as follows:

```
def decrypt(metadata, sframe_ciphertext):
  KID, CTR, header, ciphertext = parse_ciphertext(sframe_ciphertext)

  sframe_key, sframe_salt = key_store[KID]

  ctr = encode_big_endian(CTR, AEAD.Nn)
  nonce = xor(sframe_salt, ctr)
  aad = header + metadata

  return AEAD.Decrypt(sframe_key, nonce, aad, ciphertext)
```

If a ciphertext fails to decrypt because there is no key available for the KID in the SFrame header, the client MAY buffer the ciphertext and retry decryption once a key with that KID is received. If a ciphertext fails to decrypt for any other reason, the client MUST discard the ciphertext. Invalid ciphertexts SHOULD be discarded in a way that is indistinguishable (to an external observer) from having processed a valid ciphertext. In other words, the SFrame decrypt operation should be constant-time, regardless of whether decryption succeeds or fails.

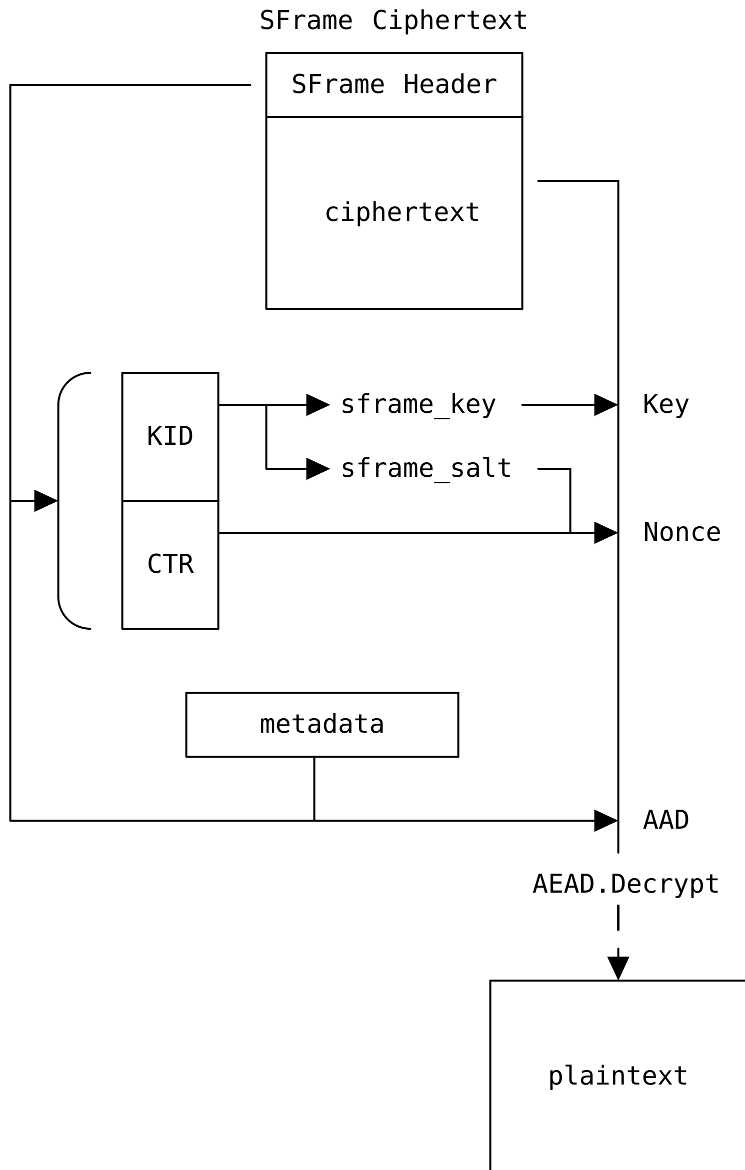


Figure 5: Decrypting an SFrame Ciphertext

4.5. Cipher Suites

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

* A hash function used for key derivation
* An AEAD encryption algorithm [RFC5116] used for frame encryption, optionally with a truncated authentication tag

This document defines the following cipher suites, with the constants defined in Section 4.4:

Name	Nh	Nka	Nk	Nn	Nt
AES_128_CTR_HMAC_SHA256_80	32	16	48	12	10
AES_128_CTR_HMAC_SHA256_64	32	16	48	12	8
AES_128_CTR_HMAC_SHA256_32	32	16	48	12	4
AES_128_GCM_SHA256_128	32	n/a	16	12	16
AES_256_GCM_SHA512_128	64	n/a	32	12	16

Table 1: SFrame cipher suite constants

Numeric identifiers for these cipher suites are defined in the IANA registry created in Section 8.1.

In the suite names, the length of the authentication tag is indicated by the last value: "_128" indicates a hundred-twenty-eight-bit tag, "_80" indicates an eighty-bit tag, "_64" indicates a sixty-four-bit tag and "_32" indicates a thirty-two-bit tag.

In a session that uses multiple media streams, different cipher suites might be configured for different media streams. For example, in order to conserve bandwidth, a session might use a cipher suite with eighty-bit tags for video frames and another cipher suite with thirty-two-bit tags for audio frames.

4.5.1. AES-CTR with SHA2

In order to allow very short tag sizes, we define a synthetic AEAD function using the authenticated counter mode of AES together with HMAC for authentication. We use an encrypt-then-MAC approach, as in SRTP [RFC3711].

Before encryption or decryption, encryption and authentication subkeys are derived from the single AEAD key. The overall length of the AEAD key is Nka + Nh, where Nka represents the key size for the AES block cipher in use and Nh represents the output size of the hash function (as in Table 2). The encryption subkey comprises the first Nka bytes and the authentication subkey comprises the remaining Nh bytes.

```
def derive_subkeys(sframe_key):
  # The encryption key comprises the first Nka bytes
  enc_key = sframe_key[..Nka]

  # The authentication key comprises Nh remaining bytes
  auth_key = sframe_key[Nka..]

  return enc_key, auth_key
```

The AEAD encryption and decryption functions are then composed of individual calls to the CTR encrypt function and HMAC. The resulting

MAC value is truncated to a number of bytes Nt fixed by the cipher suite.

```
def truncate(tag, n):
  # Take the first `n` bytes of `tag`
  return tag[..n]

def compute_tag(auth_key, nonce, aad, ct):
  aad_len = encode_big_endian(len(aad), 8)
  ct_len = encode_big_endian(len(ct), 8)
  tag_len = encode_big_endian(Nt, 8)
  auth_data = aad_len + ct_len + tag_len + nonce + aad + ct
  tag = HMAC(auth_key, auth_data)
  return truncate(tag, Nt)

def AEAD.Encrypt(key, nonce, aad, pt):
  enc_key, auth_key = derive_subkeys(key)
  initial_counter = nonce + 0x00000000 # append four zero bytes
  ct = AES-CTR.Encrypt(enc_key, initial_counter, pt)
  tag = compute_tag(auth_key, nonce, aad, ct)
  return ct + tag

def AEAD.Decrypt(key, nonce, aad, ct):
  inner_ct, tag = split_ct(ct, tag_len)

  enc_key, auth_key = derive_subkeys(key)
  candidate_tag = compute_tag(auth_key, nonce, aad, inner_ct)
  if !constant_time_equal(tag, candidate_tag):
    raise Exception("Authentication Failure")

  initial_counter = nonce + 0x00000000 # append four zero bytes
  return AES-CTR.Decrypt(enc_key, initial_counter, inner_ct)
```

5. Key Management

SFrame must be integrated with an E2E key management framework to exchange and rotate the keys used for SFrame encryption. The key management framework provides the following functions:

* Provisioning KID / base_key mappings to participating clients

* Updating the above data as clients join or leave

It is the responsibility of the application to provide the key management framework, as described in Section 9.2.

5.1. Sender Keys

If the participants in a call have a pre-existing E2E-secure channel, they can use it to distribute SFrame keys. Each client participating

in a call generates a fresh base_key value that it will use to encrypt media. The client then uses the E2E-secure channel to send their encryption key to the other participants.

In this scheme, it is assumed that receivers have a signal outside of SFrame for which client has sent a given frame (e.g., an RTP SSRC). SFrame KID values are then used to distinguish between versions of the sender's base_key.

Key IDs in this scheme have two parts: a "key generation" and a "ratchet step". Both are unsigned integers that begin at zero. The key generation increments each time the sender distributes a new key to receivers. The "ratchet step" is incremented each time the sender ratchets their key forward for forward secrecy:

```
base_key[i+1] = HKDF-Expand(
                  HKDF-Extract("", base_key[i]),
                  "SFrame 1.0 Ratchet", CipherSuite.Nh)
```

For compactness, we do not send the whole ratchet step. Instead, we send only its low-order R bits, where R is a value set by the application. Different senders may use different values of R, but each receiver of a given sender needs to know what value of R is used by the sender so that they can recognize when they need to ratchet (vs. expecting a new key). R effectively defines a re-ordering window, since no more than 2^R ratchet steps can be active at a given time. The key generation is sent in the remaining 64 - R bits of the key ID.

```
KID = (key_generation << R) + (ratchet_step % (1 << R))
```

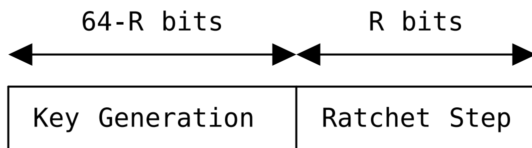


Figure 6: Structure of a KID in the Sender Keys scheme

The sender signals such a ratchet step update by sending with a KID value in which the ratchet step has been incremented. A receiver who receives from a sender with a new KID computes the new key as above. The old key may be kept for some time to allow for out-of-order delivery, but should be deleted promptly.

If a new participant joins in the middle of a session, they will need to receive from each sender (a) the current sender key for that sender and (b) the current KID value for the sender. Evicting a

participant requires each sender to send a fresh sender key to all receivers.

It is up to the application to decide when sender keys are updated. A sender key may be updated by sending a new base_key (updating the key generation) or by hashing the current base_key (updating the ratchet step). Ratcheting the key forward is useful when adding new receivers to an SFrame-based interaction, since it ensures that the new receivers can't decrypt any media encrypted before they were added. If a sender wishes to assure the opposite property when removing a receiver (i.e., ensuring that the receiver can't decrypt media after they are removed), then the sender will need to distribute a new sender key.

5.2. MLS

The Messaging Layer Security (MLS) protocol provides group authenticated key exchange [MLS-ARCH] [MLS-PROTO]. In principle, it could be used to instantiate the sender key scheme above, but it can also be used more efficiently directly.

MLS creates a linear sequence of keys, each of which is shared among the members of a group at a given point in time. When a member joins or leaves the group, a new key is produced that is known only to the augmented or reduced group. Each step in the lifetime of the group is known as an "epoch", and each member of the group is assigned an "index" that is constant for the time they are in the group.

To generate keys and nonces for SFrame, we use the MLS exporter function to generate a base_key value for each MLS epoch. Each member of the group is assigned a set of KID values, so that each member has a unique sframe_key and sframe_salt that it uses to encrypt with. Senders may choose any KID value within their assigned set of KID values, e.g., to allow a single sender to send multiple uncoordinated outbound media streams.

```
base_key = MLS-Exporter("SFrame 1.0 Base Key", "", AEAD.Nk)
```

For compactness, we do not send the whole epoch number. Instead, we send only its low-order E bits, where E is a value set by the application. E effectively defines a re-ordering window, since no more than 2^E epochs can be active at a given time. Receivers MUST be prepared for the epoch counter to roll over, removing an old epoch when a new epoch with the same E lower bits is introduced.

Let S be the number of bits required to encode a member index in the group, i.e., the smallest value such that group_size <= (1 << S). The sender index is encoded in the S bits above the epoch. The remaining 64 - S - E bits of the KID value are a context value chosen by the sender (context value 0 will produce the shortest encoded KID).

```
KID = (context << (S + E)) + (sender_index << E) + (epoch % (1 << E))
```

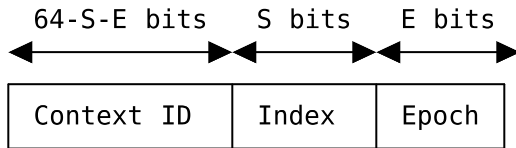


Figure 7: Structure of a KID for an MLS Sender

Once an SFrame stack has been provisioned with the sframe_epoch_secret for an epoch, it can compute the required KID values on demand (as well as the resulting SFrame keys/nonces derived from the base_key and KID), as it needs to encrypt or decrypt for a given member.

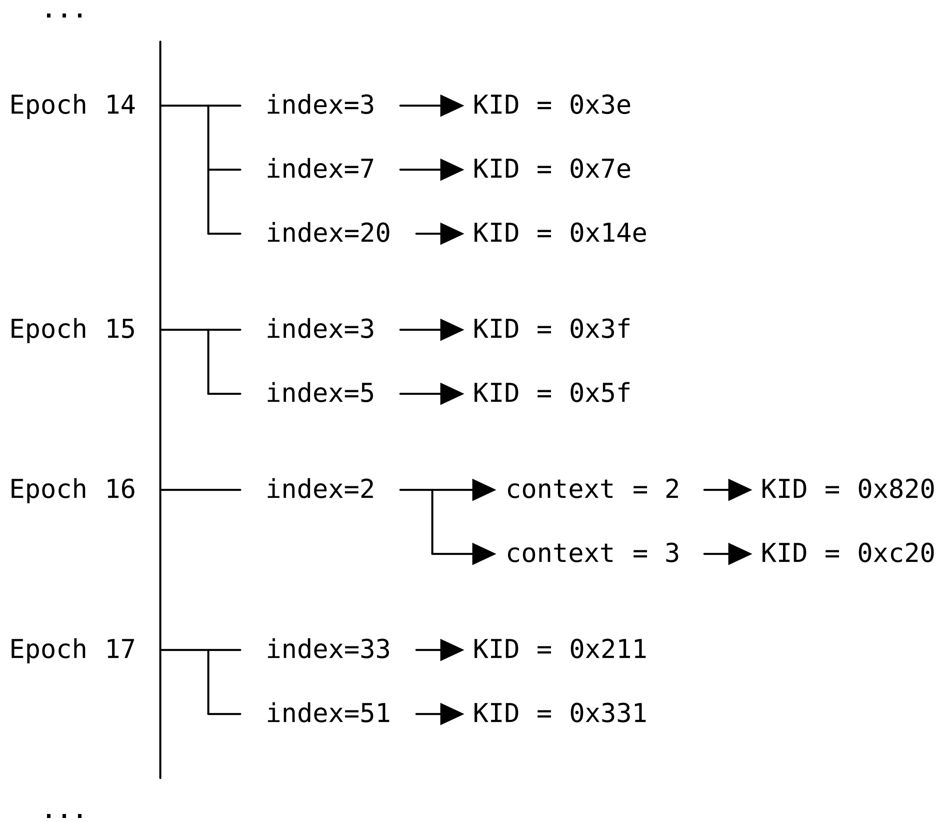


Figure 8: An example sequence of KIDs for an MLS-based SFrame session (E=4; S=6, allowing for 64 group members)

6. Media Considerations

6.1. Selective Forwarding Units

Selective Forwarding Units (SFUs) (e.g., those described in Section 3.7 of [RFC7667]) receive the media streams from each participant and select which ones should be forwarded to each of the other participants. There are several approaches for stream selection, but in general, 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 interact 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 avoid the overhead involved in establishing new transport streams, the SFU may decide to reuse previously existing streams or even pre-allocate a predefined number of streams and choose in each moment in time which participant media will be sent through it.

This means that in the same transport-level stream (e.g., an 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 in 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), a mechanism based on digital signature would be required. SFrame does not protect against such attacks.

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 the SFU selectively

removing layers, the sender MUST encapsulate each layer in a different SFrame ciphertext.

6.2. Video Key Frames

Forward Security and Post-Compromise Security require that the E2EE keys (base keys) are updated any time a participant joins or leaves the call.

The key exchange happens asynchronously 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 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.

The new Receiver will then re-request a key frame, 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 after the new E2EE key is in use, the time required for the new participant to display the video is minimized.

Note that this issue does not arise for media streams that do not have dependencies among frames, e.g., audio streams. In these streams, each frame is independently decodeable, so there is never a need to process two frames together which might be on two sides of a key rotation.

6.3. Partial Decoding

Some codecs support partial decoding, where individual packets can be decoded without waiting for the full frame to arrive. When SFrame is applied per-frame, this won't be possible because the decoder cannot access data until an entire frame has arrived and has been decrypted.

7. Security Considerations

7.1. No Header Confidentiality

SFrame provides integrity protection to the SFrame Header (the key ID and counter values), but does not provide confidentiality protection. Parties that can observe the SFrame header may learn, for example, which parties are sending SFrame payloads (from KID values) and at what rates (from CTR values). In cases where SFrame is used for end-to-end security on top of hop-by-hop protections (e.g., running over SRTP as described in Appendix C.5), the hop-by-hop security

mechanisms provide confidentiality protection of the SFrame header between hops.

7.2. No Per-Sender Authentication

SFrame does not provide per-sender authentication of media data. Any sender in a session can send media that will be associated with any other sender. This is because SFrame uses symmetric encryption to protect media data, so that any receiver also has the keys required to encrypt packets for the sender.

7.3. Key Management

Key exchange mechanism is out of scope of this document, however every client SHOULD change their keys when new clients joins or leaves the call for forward secrecy and post compromise security.

7.4. Replay

The handling of replay is out of the scope of this document. However, senders MUST reject requests to encrypt multiple times with the same key and nonce, since several AEAD algorithms fail badly in such cases (see, e.g., Section 5.1.1 of [RFC5116]).

7.5. Risks due to Short Tags

The SFrame ciphersuites based on AES-CTR allow for the use of short authentication tags, which bring a higher risk that an attacker will be able to cause an SFrame receiver to accept an SFrame ciphertext of the attacker's choosing.

Assuming that the authentication properties of the ciphersuite are robust, the only attack that an attacker can mount is an attempt to find an acceptable (ciphertext, tag) combination through brute force. Such a brute-force attack will have an expected success rate of the following form:

attacker_success_rate = attempts_per_second / 2^(8*Nt)

For example, a gigabit ethernet connection is able to transmit roughly 2^20 packets per second. If an attacker saturated such a link with guesses against a 32-bit authentication tag (Nt=4), then the attacker would succeed on average roughly once every 2^12 seconds, or about once an hour.

In a typical SFrame usage in a real-time media application, there are a few approaches to mitigating this risk:

* Receivers only accept SFrame ciphertexts over HBH-secure channels (e.g., SRTP security associations or QUIC connections). If this is

the case, only an entity that is part of such a channel can mount the above attack.

* The expected packet rate for a media stream is very predictable (and typically far lower than the above example). On the one hand, attacks at this rate will succeed even less often than the high-rate attack described above. On the other hand, the application may use an elevated packet arrival rate as a signal of a brute-force attack. This latter approach is common in other settings, e.g., mitigating brute-force attacks on passwords.

* Media applications typically do not provide feedback to media senders as to which media packets failed to decrypt. When media quality feedback mechanisms are used, decryption failures will typically appear as packet losses, but only at an aggregate level.

* Anti-replay mechanisms (see Section 7.4) prevent the attacker from re-using valid ciphertexts (either observed or guessed by the attacker). A receiver applying anti-replay controls will only accept one valid plaintext per CTR value. Since the CTR value is covered by SFrame authentication, an attacker has to do a fresh search for a valid tag for every forged ciphertext, even if the encrypted content is unchanged. In other words, when the above brute force attack succeeds, it only allows the attacker to send a single SFrame ciphertext; the ciphertext cannot be reused because either it will have the same CTR value and be discarded as a replay, or else it will have a different CTR value its tag will no longer be valid.

Nonetheless, without these mitigations, an application that makes use of short tags will be at heightened risk of forgery attacks. In many cases, it is simpler to use full-size tags and tolerate slightly higher bandwidth usage rather than add the additional defenses necessary to safely use short tags.

8. IANA Considerations

This document requests the creation of the following new IANA registry:

* SFrame Cipher Suites (Section 8.1)

This registry should be under a heading of "SFrame", and assignments are made via the Specification Required policy [RFC8126].

RFC EDITOR: Please replace XXXX throughout with the RFC number assigned to this document

8.1. SFrame Cipher Suites

This registry lists identifiers for SFrame cipher suites, as defined in Section 4.5. The cipher suite field is two bytes wide, so the valid cipher suites are in the range 0x0000 to 0xFFFF.

Template:

* Value: The numeric value of the cipher suite

* Name: The name of the cipher suite

* Recommended: Whether support for this cipher suite is recommended by the IETF. Valid values are "Y", "N", and "D", as described in Section 17.1 of [MLS-PROTO]. The default value of the "Recommended" column is "N". Setting the Recommended item to "Y" or "D", or changing an item whose current value is "Y" or "D", requires Standards Action [RFC8126].

* Reference: The document where this cipher suite is defined

Initial contents:

Value	Name	R	Reference
0x0000	Reserved	-	RFC XXXX
0x0001	AES_128_CTR_HMAC_SHA256_80	Y	RFC XXXX
0x0002	AES_128_CTR_HMAC_SHA256_64	Y	RFC XXXX
0x0003	AES_128_CTR_HMAC_SHA256_32	Y	RFC XXXX
0x0004	AES_128_GCM_SHA256_128	Y	RFC XXXX
0x0005	AES_256_GCM_SHA512_128	Y	RFC XXXX
0xF000 - 0xFFFF	Reserved for private use	-	RFC XXXX

Table 2: SFrame cipher suites

9. Application Responsibilities

To use SFrame, an application needs to define the inputs to the SFrame encryption and decryption operations, and how SFrame ciphertexts are delivered from sender to receiver (including any fragmentation and reassembly). In this section, we lay out additional requirements that an integration must meet in order for SFrame to operate securely.

In general, an application using SFrame is responsible for configuring SFrame. The application must first define when SFrame is applied at all. When SFrame is applied, the application must define which cipher suite is to be used. If new versions of SFrame are defined in the future, it will be up to the application to determine which version should be used.

This division of responsibilities is similar to the way other media parameters (e.g., codecs) are typically handled in media applications, in the sense that they are set up in some signaling protocol, and then not described in the media. Applications might find it useful to extend the protocols used for negotiating other media parameters (e.g., SDP [RFC8866]) to also negotiate parameters for SFrame.

9.1. Header Value Uniqueness

Applications MUST ensure that each (base_key, KID, CTR) combination is used for at most one SFrame encryption operation. This ensures that the (key, nonce) pairs used by the underlying AEAD algorithm are never reused. Typically this is done by assigning each sender a KID or set of KIDs, then having each sender use the CTR field as a monotonic counter, incrementing for each plaintext that is encrypted. In addition to its simplicity, this scheme minimizes overhead by keeping CTR values as small as possible.

In applications where an SFrame context might be written to persistent storage, this context needs to include the last used CTR value. When the context is used later, the application should use the stored CTR value to determine the next CTR value to be used in an encryption operation, and then write the next CTR value back to storage before using the CTR value for encryption. Storing the CTR value before usage (vs. after) helps ensure that a storage failure will not cause reuse of the same (base_key, KID, CTR) combination.

9.2. Key Management Framework

It is up to the application to provision SFrame with a mapping of KID values to base_key values and the resulting keys and salts. More importantly, the application specifies which KID values are used for which purposes (e.g., by which senders). An application's KID assignment strategy MUST be structured to assure the non-reuse properties discussed in Section 9.1.

It is also up to the application to define a rotation schedule for keys. For example, one application might have an ephemeral group for every call and keep rotating keys when end points join or leave the call, while another application could have a persistent group that can be used for multiple calls and simply derives ephemeral symmetric keys for a specific call.

It should be noted that KID values are not encrypted by SFrame, and are thus visible to any application-layer intermediaries that might handle an SFrame ciphertext. If there are application semantics included in KID values, then this information would be exposed to intermediaries. For example, in the scheme of Section 5.1, the number

of ratchet steps per sender is exposed, and in the scheme of Section 5.2, the number of epochs and the MLS sender ID of the SFrame sender are exposed.

9.3. Anti-Replay

It is the responsibility of the application to handle anti-replay. Replay by network attackers is assumed to be prevented by network-layer facilities (e.g., TLS, SRTP). As mentioned in Section 7.4, senders MUST reject requests to encrypt multiple times with the same key and nonce.

It is not mandatory to implement anti-replay on the receiver side. Receivers MAY apply time or counter based anti-replay mitigations. For example, Section 3.3.2 of [RFC3711] specifies a counter-based anti-replay mitigation, which could be adapted to use with SFrame, using the CTR field as the counter.

9.4. Metadata

The metadata input to SFrame operations is pure application-specified data. As such, it is up to the application to define what information should go in the metadata input and ensure that it is provided to the encryption and decryption functions at the appropriate points. A receiver MUST NOT use SFrame-authenticated metadata until after the SFrame decrypt function has authenticated it, unless the purpose of such usage is to prepare an SFrame ciphertext for SFrame decryption. Essentially, metadata may be used "upstream of SFrame" in a processing pipeline, but only to prepare for SFrame decryption.

For example, consider an application where SFrame is used to encrypt audio frames that are sent over SRTP, with some application data included in the RTP header extension. Suppose the application also includes this application data in the SFrame metadata, so that the SFU is allowed to read, but not modify the application data. A receiver can use the application data in the RTP header extension as part of the standard SRTP decryption process, since this is required to recover the SFrame ciphertext carried in the SRTP payload. However, the receiver MUST NOT use the application data for other purposes before SFrame decryption has authenticated the application data.

10. References

10.1. Normative References

[MLS-PROTO] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

[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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/rfc/rfc5116>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

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

10.2. Informative References

[I-D.codec-agnostic-rtp-payload-format] Murillo, S. G. and A. Gouaillard, "Codec agnostic RTP payload format for video", Work in Progress, Internet-Draft, draft-codec-agnostic-rtp-payload-format-00, 19 February 2021, <https://datatracker.ietf.org/doc/html/draft-codec-agnostic-rtp-payload-format-00>.

[I-D.ietf-moq-transport] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., and I. Swett, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-03, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-03>.

[I-D.ietf-webtrans-overview]
Vasiliev, V., "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-ietf-webtrans-overview-07, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-overview-07>.

[MLS-ARCH] Beurdouche, B., Rescorla, E., Omara, E., Inguva, S., and A. Duric, "The Messaging Layer Security (MLS) Architecture", Work in Progress, Internet-Draft, draft-ietf-mls-architecture-13, 22 March 2024, <https://

datatracker.ietf.org/doc/html/draft-ietf-mls-architecture-13>.

[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/rfc/rfc3711>.

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <https://www.rfc-editor.org/rfc/rfc6716>.

[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>.

[RFC7667] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 7667, DOI 10.17487/RFC7667, November 2015, <https://www.rfc-editor.org/rfc/rfc7667>.

[RFC8723] Jennings, C., Jones, P., Barnes, R., and A.B. Roach, "Double Encryption Procedures for the Secure Real-Time Transport Protocol (SRTP)", RFC 8723, DOI 10.17487/RFC8723, April 2020, <https://www.rfc-editor.org/rfc/rfc8723>.

[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>.

[TestVectors] "SFrame Test Vectors", 2023, <https://github.com/eomara/sframe/blob/master/test-vectors.json>.

Appendix A. Acknowledgements

The authors wish to specially thank Dr. Alex Gouaillard as one of the early contributors to the document. His passion and energy were key to the design and development of SFrame.

Appendix B. Example API

This section is not normative.

This section describes a notional API that an SFrame implementation might expose. The core concept is an "SFrame context", within which KID values are meaningful. In the key management scheme described in

Section 5.1, each sender has a different context; in the scheme described in Section 5.2, all senders share the same context.

An SFrame context stores mappings from KID values to "key contexts", which are different depending on whether the KID is to be used for sending or receiving (an SFrame key should never be used for both operations). A key context tracks the key and salt associated to the KID, and the current CTR value. A key context to be used for sending also tracks the next CTR value to be used.

The primary operations on an SFrame context are as follows:

* **Create an SFrame context:** The context is initialized with a ciphersuite and no KID mappings.

* **Adding a key for sending:** The key and salt are derived from the base key, and used to initialize a send context, together with a zero counter value.

* **Adding a key for receiving:** The key and salt are derived from the base key, and used to initialize a send context.

* **Encrypt a plaintext:** Encrypt a given plaintext using the key for a given KID, including the specified metadata.

* **Decrypt an SFrame ciphertext:** Decrypt an SFrame ciphertext with the KID and CTR values specified in the SFrame Header, and the provided metadata.

Figure 9 shows an example of the types of structures and methods that could be used to create an SFrame API in Rust.

```
type KeyId = u64;
type Counter = u64;
type CipherSuite = u16;

struct SendKeyContext {
  key: Vec<u8>,
  salt: Vec<u8>,
  next_counter: Counter,
}

struct RecvKeyContext {
  key: Vec<u8>,
  salt: Vec<u8>,
}

struct SFrameContext {
  cipher_suite: CipherSuite,
  send_keys: HashMap<KeyId, SendKeyContext>,
  recv_keys: HashMap<KeyId, RecvKeyContext>,
}

trait SFrameContextMethods {
  fn create(cipher_suite: CipherSuite) -> Self;
  fn add_send_key(&self, kid: KeyId, base_key: &[u8]);
  fn add_recv_key(&self, kid: KeyId, base_key: &[u8]);
  fn encrypt(&mut self, kid: KeyId, metadata: &[u8],
             plaintext: &[u8]) -> Vec<u8>;
  fn decrypt(&self, metadata: &[u8], ciphertext: &[u8]) -> Vec<u8>;
}
```

Figure 9: An Example SFrame API

Appendix C. Overhead Analysis

Any use of SFrame will impose overhead in terms of the amount of bandwidth necessary to transmit a given media stream. Exactly how much overhead will be added depends on several factors:

* How many senders are involved in a conference (length of KID)

* How long the conference has been going on (length of CTR)

* The cipher suite in use (length of authentication tag)

* Whether SFrame is used to encrypt packets, whole frames, or some other unit

Overall, the overhead rate in kilobits per second can be estimated as:

```
OverheadKbps = (1 + |CTR| + |KID| + |TAG|) * 8 * CTPerSecond / 1024
```

Here the constant value 1 reflects the fixed SFrame header; |CTR| and |KID| reflect the lengths of those fields; |TAG| reflects the cipher overhead; and CTPerSecond reflects the number of SFrame ciphertexts sent per second (e.g., packets or frames per second).

In the remainder of this secton, we compute overhead estimates for a collection of common scenarios.

C.1. Assumptions

In the below calculations, we make conservative assumptions about SFrame overhead, so that the overhead amounts we compute here are likely to be an upper bound on those seen in practice.

Field	Bytes	Explanation
Fixed header	1	Fixed
Key ID (KID)	2	>255 senders; or MLS epoch (E=4) and >16 senders
Counter (CTR)	3	More than 24 hours of media in common cases
Cipher overhead	16	Full GCM tag (longest defined here)

Table 3

In total, then, we assume that each SFrame encryption will add 22 bytes of overhead.

We consider two scenarios, applying SFrame per-frame and per-packet. In each scenario, we compute the SFrame overhead in absolute terms (Kbps) and as a percentage of the base bandwidth.

C.2. Audio

In audio streams, there is typically a one-to-one relationship between frames and packets, so the overhead is the same whether one uses SFrame at a per-packet or per-frame level.

The below table considers three scenarios, based on recommended configurations of the Opus codec [RFC6716]:

* Narrow-band speech: 120ms packets, 8Kbps

* Full-band speech: 20ms packets, 32Kbps

* Full-band stereo music: 10ms packets, 128Kbps

Scenario	fps	Base Kbps	Overhead Kbps	Overhead %
NB speech, 120ms packets	8.3	8	1.4	17.9%

Scenario	fps	Base Kbps	Overhead Kbps	Overhead %
FB speech, 20ms packets	50	32	8.6	26.9%
FB stereo, 10ms packets	100	128	17.2	13.4%

Table 4: SFrame overhead for audio streams

C.3. Video

Video frames can be larger than an MTU and thus are commonly split across multiple frames. Table 5 and Table 6 show the estimated overhead of encrypting a video stream, where SFrame is applied per-frame and per-packet, respectively. The choices of resolution, frames per second, and bandwidth are chosen to roughly reflect the capabilities of modern video codecs across a range from very low to very high quality.

Scenario	fps	Base Kbps	Overhead Kbps	Overhead %
426 x 240	7.5	45	1.3	2.9%
640 x 360	15	200	2.6	1.3%
640 x 360	30	400	5.2	1.3%
1280 x 720	30	1500	5.2	0.3%
1920 x 1080	60	7200	10.3	0.1%

Table 5: SFrame overhead for a video stream encrypted per-frame

Scenario	fps	pps	Base Kbps	Overhead Kbps	Overhead %
426 x 240	7.5	7.5	45	1.3	2.9%
640 x 360	15	30	200	5.2	2.6%
640 x 360	30	60	400	10.3	2.6%
1280 x 720	30	180	1500	30.9	2.1%
1920 x 1080	60	780	7200	134.1	1.9%

Table 6: SFrame overhead for a video stream encrypted per-packet

In the per-frame case, the SFrame percentage overhead approaches zero as the quality of the video goes up, since bandwidth is driven more by picture size than frame rate. In the per-packet case, the SFrame percentage overhead approaches the ratio between the SFrame overhead per packet and the MTU (here 22 bytes of SFrame overhead divided by an assumed 1200-byte MTU, or about 1.8%).

C.4. Conferences

Real conferences usually involve several audio and video streams. The overhead of SFrame in such a conference is the aggregate of the overhead over all the individual streams. Thus, while SFrame incurs a large percentage overhead on an audio stream, if the conference also involves a video stream, then the audio overhead is likely negligible relative to the overall bandwidth of the conference.

For example, Table 7 shows the overhead estimates for a two person conference where one person is sending low-quality media and the other sending high-quality. (And we assume that SFrame is applied per-frame.) The video streams dominate the bandwidth at the SFU, so the total bandwidth overhead is only around 1%.

Stream	Base Kbps	Overhead Kbps	Overhead %
Participant 1 audio	8	1.4	17.9%
Participant 1 video	45	1.3	2.9%
Participant 2 audio	32	9	26.9%
Participant 2 video	1500	5	0.3%
Total at SFU	1585	16.5	1.0%

Table 7: SFrame overhead for a two-person conference

C.5. SFrame over RTP

SFrame is a generic encapsulation format, but many of the applications in which it is likely to be integrated are based on RTP. This section discusses how an integration between SFrame and RTP could be done, and some of the challenges that would need to be overcome.

As discussed in Section 4.1, there are two natural patterns for integrating SFrame into an application: applying SFrame per-frame or per-packet. In RTP-based applications, applying SFrame per-packet means that the payload of each RTP packet will be an SFrame ciphertext, starting with an SFrame Header, as shown in Figure 10. Applying SFrame per-frame means that different RTP payloads will have different formats: The first payload of a frame will contain the SFrame headers, and subsequent payloads will contain further chunks of the ciphertext, as shown in Figure 11.

In order for these media payloads to be properly interpreted by receivers, receivers will need to be configured to know which of the above schemes the sender has applied to a given sequence of RTP packets. SFrame does not provide a mechanism for distributing this configuration information. In applications that use SDP for negotiating RTP media streams [RFC8866], an appropriate extension to SDP could provide this function.

Applying SFrame per-frame also requires that packetization and depacketization be done in a generic manner that does not depend on the media content of the packets, since the content being packetized / depacketized will be opaque ciphertext (except for the SFrame header). In order for such a generic packetization scheme to work interoperably one would have to be defined, e.g., as proposed in [I-D.codec-agnostic-rtp-payload-format].

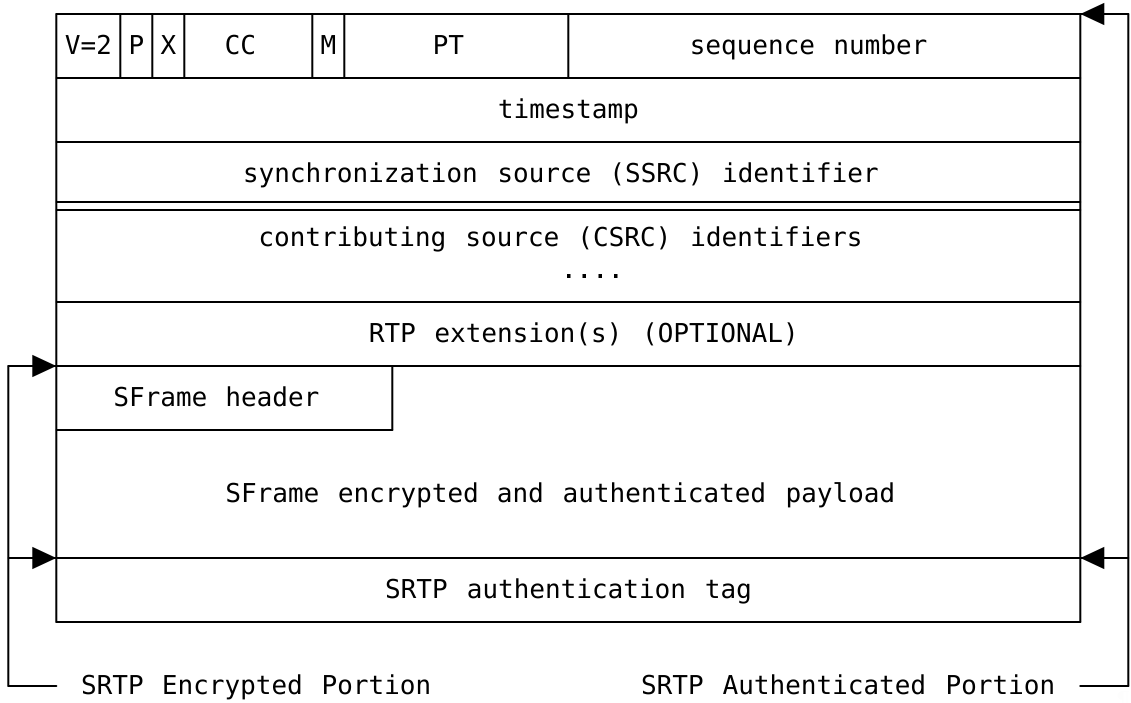


Figure 10: SRTP packet with SFrame-protected payload

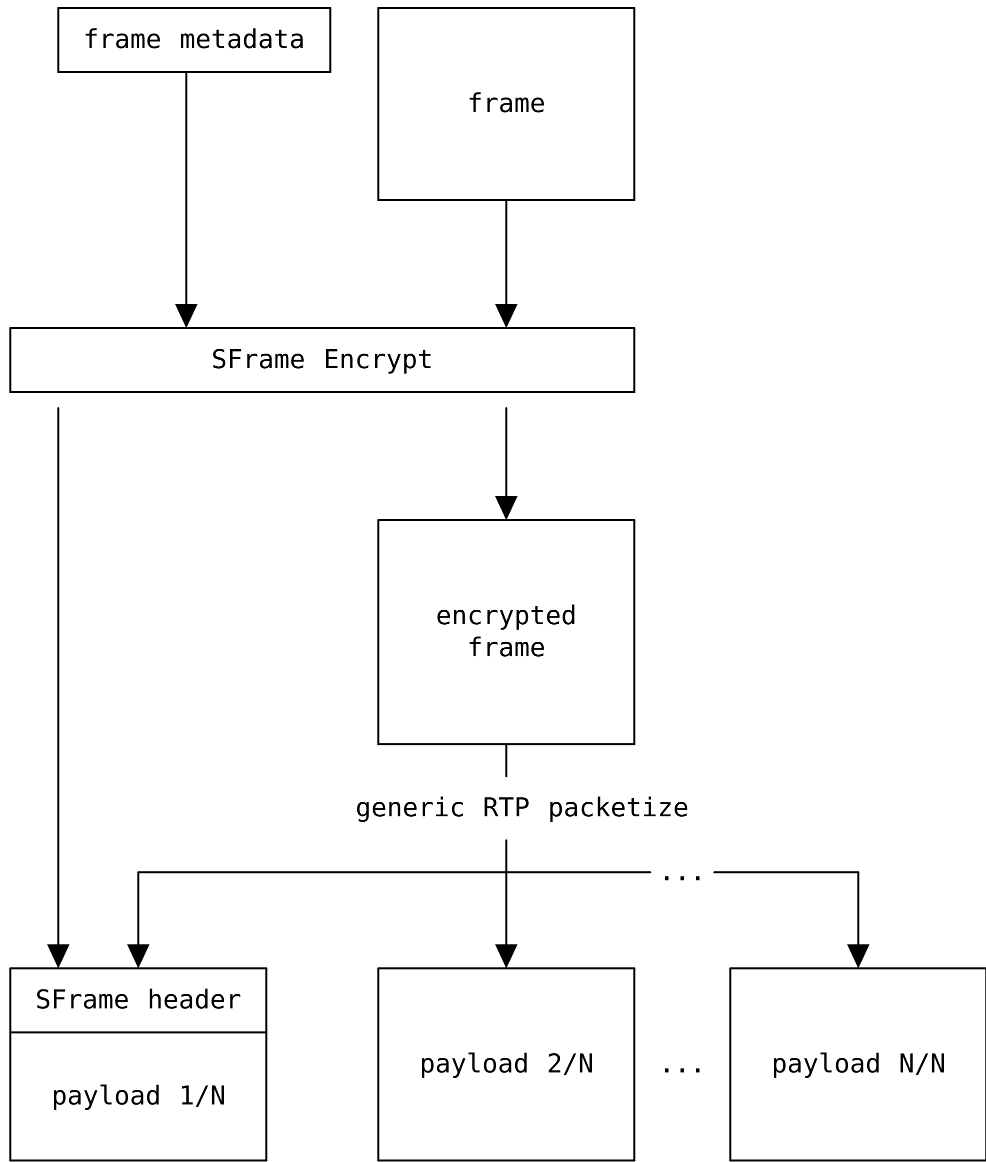


Figure 11: Encryption flow with per-frame encryption for RTP

Appendix D. Test Vectors

This section provides a set of test vectors that implementations can use to verify that they correctly implement SFrame encryption and decryption. In addition to test vectors for the overall process of SFrame encryption/decryption, we also provide test vectors for header encoding/decoding, and for AEAD encryption/decryption using the AES-CTR construction defined in Section 4.5.1.

All values are either numeric or byte strings. Numeric values are represented as hex values, prefixed with 0x. Byte strings are represented in hex encoding.

Line breaks and whitespace within values are inserted to conform to the width requirements of the RFC format. They should be removed before use.

These test vectors are also available in JSON format at [TestVectors]. In the JSON test vectors, numeric values are JSON numbers and byte string values are JSON strings containing the hex encoding of the byte strings.

D.1. Header encoding/decoding

For each case, we provide:

* kid: A KID value
* ctr: A CTR value
* header: An encoded SFrame header

An implementation should verify that:

* Encoding a header with the KID and CTR results in the provided header value
* Decoding the provided header value results in the provided KID and CTR values

```
kid: 0x0000000000000000
ctr: 0x0000000000000000
header: 00

kid: 0x0000000000000000
ctr: 0x0000000000000001
header: 01

kid: 0x0000000000000000
ctr: 0x00000000000000ff
header: 08ff

kid: 0x0000000000000000
ctr: 0x0000000000000100
header: 090100

kid: 0x0000000000000000
ctr: 0x000000000000ffff
header: 09ffff

kid: 0x0000000000000000
ctr: 0x0000000000010000
header: 0a010000
```

```
kid: 0x0000000000000000
ctr: 0x0000000000ffffff
header: 0affffff

kid: 0x0000000000000000
ctr: 0x0000000001000000
header: 0b01000000

kid: 0x0000000000000000
ctr: 0x00000000ffffffff
header: 0bffffffff

kid: 0x0000000000000000
ctr: 0x0000000100000000
header: 0c0100000000

kid: 0x0000000000000000
ctr: 0x000000ffffffffff
header: 0cffffffffff

kid: 0x0000000000000000
ctr: 0x0000010000000000
header: 0d010000000000

kid: 0x0000000000000000
ctr: 0x0000ffffffffffff
header: 0dffffffffffff

kid: 0x0000000000000000
ctr: 0x0001000000000000
header: 0e01000000000000

kid: 0x0000000000000000
ctr: 0x00ffffffffffffff
header: 0effffffffffffff

kid: 0x0000000000000000
ctr: 0x0100000000000000
header: 0f0100000000000000

kid: 0x0000000000000000
ctr: 0xffffffffffffffff
header: 0fffffffffffffffff

kid: 0x0000000000000001
ctr: 0x0000000000000000
header: 10
```

```
kid: 0x0000000000000001
ctr: 0x0000000000000001
header: 11

kid: 0x0000000000000001
ctr: 0x00000000000000ff
header: 18ff

kid: 0x0000000000000001
ctr: 0x0000000000000100
header: 190100

kid: 0x0000000000000001
ctr: 0x000000000000ffff
header: 19ffff

kid: 0x0000000000000001
ctr: 0x0000000000010000
header: 1a010000

kid: 0x0000000000000001
ctr: 0x0000000000ffffff
header: 1affffff

kid: 0x0000000000000001
ctr: 0x0000000001000000
header: 1b01000000

kid: 0x0000000000000001
ctr: 0x00000000ffffffff
header: 1bffffffff

kid: 0x0000000000000001
ctr: 0x0000000100000000
header: 1c0100000000

kid: 0x0000000000000001
ctr: 0x000000ffffffffff
header: 1cffffffffff

kid: 0x0000000000000001
ctr: 0x0000010000000000
header: 1d010000000000

kid: 0x0000000000000001
ctr: 0x0000ffffffffffff
header: 1dffffffffffff
```

```
kid: 0x0000000000000001
ctr: 0x0001000000000000
header: 1e01000000000000

kid: 0x0000000000000001
ctr: 0x00ffffffffffffff
header: 1effffffffffffff

kid: 0x0000000000000001
ctr: 0x0100000000000000
header: 1f0100000000000000

kid: 0x0000000000000001
ctr: 0xffffffffffffffff
header: 1fffffffffffffffff

kid: 0x00000000000000ff
ctr: 0x0000000000000000
header: 80ff

kid: 0x00000000000000ff
ctr: 0x0000000000000001
header: 81ff

kid: 0x00000000000000ff
ctr: 0x00000000000000ff
header: 88ffff

kid: 0x00000000000000ff
ctr: 0x0000000000000100
header: 89ff0100

kid: 0x00000000000000ff
ctr: 0x000000000000ffff
header: 89ffffff

kid: 0x00000000000000ff
ctr: 0x0000000000010000
header: 8aff010000

kid: 0x00000000000000ff
ctr: 0x0000000000ffffff
header: 8affffffff

kid: 0x00000000000000ff
ctr: 0x0000000001000000
header: 8bff01000000
```

```
kid: 0x00000000000000ff
ctr: 0x00000000ffffffff
header: 8bffffffffff

kid: 0x00000000000000ff
ctr: 0x0000000100000000
header: 8cff0100000000

kid: 0x00000000000000ff
ctr: 0x000000ffffffffff
header: 8cffffffffffff

kid: 0x00000000000000ff
ctr: 0x0000010000000000
header: 8dff010000000000

kid: 0x00000000000000ff
ctr: 0x0000ffffffffffff
header: 8dffffffffffffff

kid: 0x00000000000000ff
ctr: 0x0001000000000000
header: 8eff01000000000000

kid: 0x00000000000000ff
ctr: 0x00ffffffffffffff
header: 8effffffffffffffff

kid: 0x00000000000000ff
ctr: 0x0100000000000000
header: 8fff0100000000000000

kid: 0x00000000000000ff
ctr: 0xffffffffffffffff
header: 8fffffffffffffffffff

kid: 0x0000000000000100
ctr: 0x0000000000000000
header: 900100

kid: 0x0000000000000100
ctr: 0x0000000000000001
header: 910100

kid: 0x0000000000000100
ctr: 0x00000000000000ff
header: 980100ff
```

```
kid: 0x0000000000000100
ctr: 0x0000000000000100
header: 9901000100

kid: 0x0000000000000100
ctr: 0x000000000000ffff
header: 990100ffff

kid: 0x0000000000000100
ctr: 0x0000000000010000
header: 9a0100010000

kid: 0x0000000000000100
ctr: 0x0000000000ffffff
header: 9a0100ffffff

kid: 0x0000000000000100
ctr: 0x0000000001000000
header: 9b010001000000

kid: 0x0000000000000100
ctr: 0x00000000ffffffff
header: 9b0100ffffffff

kid: 0x0000000000000100
ctr: 0x0000000100000000
header: 9c01000100000000

kid: 0x0000000000000100
ctr: 0x000000ffffffffff
header: 9c0100ffffffffff

kid: 0x0000000000000100
ctr: 0x0000010000000000
header: 9d0100010000000000

kid: 0x0000000000000100
ctr: 0x0000ffffffffffff
header: 9d0100ffffffffffff

kid: 0x0000000000000100
ctr: 0x0001000000000000
header: 9e010001000000000000

kid: 0x0000000000000100
ctr: 0x00ffffffffffffff
header: 9e0100ffffffffffffff
```

```
kid: 0x0000000000000100
ctr: 0x0100000000000000
header: 9f01000100000000000000

kid: 0x0000000000000100
ctr: 0xffffffffffffffff
header: 9f0100ffffffffffffffff

kid: 0x000000000000ffff
ctr: 0x0000000000000000
header: 90ffff

kid: 0x000000000000ffff
ctr: 0x0000000000000001
header: 91ffff

kid: 0x000000000000ffff
ctr: 0x00000000000000ff
header: 98ffffff

kid: 0x000000000000ffff
ctr: 0x0000000000000100
header: 99ffff0100

kid: 0x000000000000ffff
ctr: 0x000000000000ffff
header: 99ffffffff

kid: 0x000000000000ffff
ctr: 0x0000000000010000
header: 9affff010000

kid: 0x000000000000ffff
ctr: 0x0000000000ffffff
header: 9affffffffff

kid: 0x000000000000ffff
ctr: 0x0000000001000000
header: 9bffff01000000

kid: 0x000000000000ffff
ctr: 0x00000000ffffffff
header: 9bffffffffffff

kid: 0x000000000000ffff
ctr: 0x0000000100000000
header: 9cffff0100000000
```

```
kid: 0x000000000000ffff
ctr: 0x000000ffffffffff
header: 9cffffffffffffff

kid: 0x000000000000ffff
ctr: 0x0000010000000000
header: 9dffff010000000000

kid: 0x000000000000ffff
ctr: 0x0000ffffffffffff
header: 9dffffffffffffffff

kid: 0x000000000000ffff
ctr: 0x0001000000000000
header: 9effff01000000000000

kid: 0x000000000000ffff
ctr: 0x00ffffffffffffff
header: 9effffffffffffffffff

kid: 0x000000000000ffff
ctr: 0x0100000000000000
header: 9fffff0100000000000000

kid: 0x000000000000ffff
ctr: 0xffffffffffffffff
header: 9fffffffffffffffffffff

kid: 0x0000000000010000
ctr: 0x0000000000000000
header: a0010000

kid: 0x0000000000010000
ctr: 0x0000000000000001
header: a1010000

kid: 0x0000000000010000
ctr: 0x00000000000000ff
header: a8010000ff

kid: 0x0000000000010000
ctr: 0x0000000000000100
header: a90100000100

kid: 0x0000000000010000
ctr: 0x000000000000ffff
header: a9010000ffff
```

kid: 0x0000000000010000
ctr: 0x0000000000010000
header: aa010000010000

kid: 0x0000000000010000
ctr: 0x0000000000ffffff
header: aa010000ffffff

kid: 0x0000000000010000
ctr: 0x0000000001000000
header: ab01000001000000

kid: 0x0000000000010000
ctr: 0x00000000ffffffff
header: ab010000ffffffff

kid: 0x0000000000010000
ctr: 0x0000000100000000
header: ac0100000100000000

kid: 0x0000000000010000
ctr: 0x000000ffffffffff
header: ac010000ffffffffff

kid: 0x0000000000010000
ctr: 0x0000010000000000
header: ad010000010000000000

kid: 0x0000000000010000
ctr: 0x0000ffffffffffff
header: ad010000ffffffffffff

kid: 0x0000000000010000
ctr: 0x0001000000000000
header: ae01000001000000000000

kid: 0x0000000000010000
ctr: 0x00ffffffffffffff
header: ae010000ffffffffffffff

kid: 0x0000000000010000
ctr: 0x0100000000000000
header: af0100000100000000000000

kid: 0x0000000000010000
ctr: 0xffffffffffffffff
header: af010000ffffffffffffffff

```
kid: 0x0000000000ffffff
ctr: 0x0000000000000000
header: a0ffffff

kid: 0x0000000000ffffff
ctr: 0x0000000000000001
header: a1ffffff

kid: 0x0000000000ffffff
ctr: 0x00000000000000ff
header: a8ffffffff

kid: 0x0000000000ffffff
ctr: 0x0000000000000100
header: a9ffffff0100

kid: 0x0000000000ffffff
ctr: 0x000000000000ffff
header: a9ffffffffff

kid: 0x0000000000ffffff
ctr: 0x0000000000010000
header: aaffffff010000

kid: 0x0000000000ffffff
ctr: 0x0000000000ffffff
header: aaffffffffffff

kid: 0x0000000000ffffff
ctr: 0x0000000001000000
header: abffffff01000000

kid: 0x0000000000ffffff
ctr: 0x00000000ffffffff
header: abffffffffffffff

kid: 0x0000000000ffffff
ctr: 0x0000000100000000
header: acffffff0100000000

kid: 0x0000000000ffffff
ctr: 0x000000ffffffffff
header: acffffffffffffffff

kid: 0x0000000000ffffff
ctr: 0x0000010000000000
header: adffffff010000000000
```

```
kid: 0x0000000000ffffff
ctr: 0x0000ffffffffffff
header: adffffffffffffffffff

kid: 0x0000000000ffffff
ctr: 0x0001000000000000
header: aeffffff01000000000000

kid: 0x0000000000ffffff
ctr: 0x00ffffffffffffff
header: aeffffffffffffffffffff

kid: 0x0000000000ffffff
ctr: 0x0100000000000000
header: afffffff0100000000000000

kid: 0x0000000000ffffff
ctr: 0xffffffffffffffff
header: afffffffffffffffffffffff

kid: 0x0000000001000000
ctr: 0x0000000000000000
header: b001000000

kid: 0x0000000001000000
ctr: 0x0000000000000001
header: b101000000

kid: 0x0000000001000000
ctr: 0x00000000000000ff
header: b801000000ff

kid: 0x0000000001000000
ctr: 0x0000000000000100
header: b9010000000100

kid: 0x0000000001000000
ctr: 0x000000000000ffff
header: b901000000ffff

kid: 0x0000000001000000
ctr: 0x0000000000010000
header: ba01000000010000

kid: 0x0000000001000000
ctr: 0x0000000000ffffff
header: ba01000000ffffff
```

```
kid: 0x0000000001000000
ctr: 0x0000000001000000
header: bb0100000001000000

kid: 0x0000000001000000
ctr: 0x00000000ffffffff
header: bb01000000ffffffff

kid: 0x0000000001000000
ctr: 0x0000000100000000
header: bc010000000100000000

kid: 0x0000000001000000
ctr: 0x000000ffffffffff
header: bc01000000ffffffffff

kid: 0x0000000001000000
ctr: 0x0000010000000000
header: bd01000000010000000000

kid: 0x0000000001000000
ctr: 0x0000ffffffffffff
header: bd01000000ffffffffffff

kid: 0x0000000001000000
ctr: 0x0001000000000000
header: be0100000001000000000000

kid: 0x0000000001000000
ctr: 0x00ffffffffffffff
header: be01000000ffffffffffffff

kid: 0x0000000001000000
ctr: 0x0100000000000000
header: bf010000000100000000000000

kid: 0x0000000001000000
ctr: 0xffffffffffffffff
header: bf01000000ffffffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0000000000000000
header: b0ffffffff

kid: 0x00000000ffffffff
ctr: 0x0000000000000001
header: b1ffffffff
```

```
kid: 0x00000000ffffffff
ctr: 0x00000000000000ff
header: b8ffffffffff

kid: 0x00000000ffffffff
ctr: 0x0000000000000100
header: b9ffffffff0100

kid: 0x00000000ffffffff
ctr: 0x000000000000ffff
header: b9ffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0000000000010000
header: baffffffff010000

kid: 0x00000000ffffffff
ctr: 0x0000000000ffffff
header: baffffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0000000001000000
header: bbffffffff01000000

kid: 0x00000000ffffffff
ctr: 0x00000000ffffffff
header: bbffffffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0000000100000000
header: bcffffffff0100000000

kid: 0x00000000ffffffff
ctr: 0x000000ffffffffff
header: bcffffffffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0000010000000000
header: bdffffffff010000000000

kid: 0x00000000ffffffff
ctr: 0x0000ffffffffffff
header: bdffffffffffffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0001000000000000
header: beffffffff01000000000000
```

```
kid: 0x00000000ffffffff
ctr: 0x00ffffffffffffff
header: beffffffffffffffffffffff

kid: 0x00000000ffffffff
ctr: 0x0100000000000000
header: bfffffffff0100000000000000

kid: 0x00000000ffffffff
ctr: 0xffffffffffffffff
header: bfffffffffffffffffffffffff

kid: 0x0000000100000000
ctr: 0x0000000000000000
header: c00100000000

kid: 0x0000000100000000
ctr: 0x0000000000000001
header: c10100000000

kid: 0x0000000100000000
ctr: 0x00000000000000ff
header: c80100000000ff

kid: 0x0000000100000000
ctr: 0x0000000000000100
header: c901000000000100

kid: 0x0000000100000000
ctr: 0x000000000000ffff
header: c90100000000ffff

kid: 0x0000000100000000
ctr: 0x0000000000010000
header: ca0100000000010000

kid: 0x0000000100000000
ctr: 0x0000000000ffffff
header: ca0100000000ffffff

kid: 0x0000000100000000
ctr: 0x0000000001000000
header: cb010000000001000000

kid: 0x0000000100000000
ctr: 0x00000000ffffffff
header: cb0100000000ffffffff
```

kid: 0x0000000100000000
ctr: 0x0000000100000000
header: cc01000000000100000000

kid: 0x0000000100000000
ctr: 0x000000ffffffffff
header: cc0100000000ffffffffff

kid: 0x0000000100000000
ctr: 0x0000010000000000
header: cd0100000000010000000000

kid: 0x0000000100000000
ctr: 0x0000ffffffffffff
header: cd0100000000ffffffffffff

kid: 0x0000000100000000
ctr: 0x0001000000000000
header: ce010000000001000000000000

kid: 0x0000000100000000
ctr: 0x00ffffffffffffff
header: ce0100000000ffffffffffffff

kid: 0x0000000100000000
ctr: 0x0100000000000000
header: cf01000000000100000000000000

kid: 0x0000000100000000
ctr: 0xffffffffffffffff
header: cf0100000000ffffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000000000000000
header: c0ffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000000000000001
header: c1ffffffffff

kid: 0x000000ffffffffff
ctr: 0x00000000000000ff
header: c8ffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000000000000100
header: c9ffffffffff0100

kid: 0x000000ffffffffff
ctr: 0x000000000000ffff
header: c9ffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000000000010000
header: caffffffffff010000

kid: 0x000000ffffffffff
ctr: 0x0000000000ffffff
header: caffffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000000001000000
header: cbffffffffff01000000

kid: 0x000000ffffffffff
ctr: 0x00000000ffffffff
header: cbffffffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000000100000000
header: ccffffffffff0100000000

kid: 0x000000ffffffffff
ctr: 0x000000ffffffffff
header: ccffffffffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0000010000000000
header: cdffffffffff010000000000

kid: 0x000000ffffffffff
ctr: 0x0000ffffffffffff
header: cdffffffffffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0001000000000000
header: ceffffffffff01000000000000

kid: 0x000000ffffffffff
ctr: 0x00ffffffffffffff
header: ceffffffffffffffffffffffff

kid: 0x000000ffffffffff
ctr: 0x0100000000000000
header: cfffffffffff0100000000000000

```
kid: 0x000000ffffffffff
ctr: 0xffffffffffffffff
header: cfffffffffffffffffffffffffff

kid: 0x0000010000000000
ctr: 0x0000000000000000
header: d0010000000000

kid: 0x0000010000000000
ctr: 0x0000000000000001
header: d1010000000000

kid: 0x0000010000000000
ctr: 0x00000000000000ff
header: d8010000000000ff

kid: 0x0000010000000000
ctr: 0x0000000000000100
header: d90100000000000100

kid: 0x0000010000000000
ctr: 0x000000000000ffff
header: d9010000000000ffff

kid: 0x0000010000000000
ctr: 0x0000000000010000
header: da010000000000010000

kid: 0x0000010000000000
ctr: 0x0000000000ffffff
header: da010000000000ffffff

kid: 0x0000010000000000
ctr: 0x0000000001000000
header: db01000000000001000000

kid: 0x0000010000000000
ctr: 0x00000000ffffffff
header: db010000000000ffffffff

kid: 0x0000010000000000
ctr: 0x0000000100000000
header: dc0100000000000100000000

kid: 0x0000010000000000
ctr: 0x000000ffffffffff
header: dc010000000000ffffffffff
```

```
kid: 0x0000010000000000
ctr: 0x0000010000000000
header: dd010000000000010000000000

kid: 0x0000010000000000
ctr: 0x0000ffffffffffff
header: dd010000000000ffffffffffff

kid: 0x0000010000000000
ctr: 0x0001000000000000
header: de01000000000001000000000000

kid: 0x0000010000000000
ctr: 0x00ffffffffffffff
header: de010000000000ffffffffffffff

kid: 0x0000010000000000
ctr: 0x0100000000000000
header: df0100000000000100000000000000

kid: 0x0000010000000000
ctr: 0xffffffffffffffff
header: df010000000000ffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000000000000000
header: d0ffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000000000000001
header: d1ffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x00000000000000ff
header: d8ffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000000000000100
header: d9ffffffffffff0100

kid: 0x0000ffffffffffff
ctr: 0x000000000000ffff
header: d9ffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000000000010000
header: daffffffffffff010000
```

```
kid: 0x0000ffffffffffff
ctr: 0x0000000000ffffff
header: daffffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000000001000000
header: dbffffffffffff01000000

kid: 0x0000ffffffffffff
ctr: 0x00000000ffffffff
header: dbffffffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000000100000000
header: dcffffffffffff0100000000

kid: 0x0000ffffffffffff
ctr: 0x000000ffffffffff
header: dcffffffffffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0000010000000000
header: ddffffffffffff010000000000

kid: 0x0000ffffffffffff
ctr: 0x0000ffffffffffff
header: ddffffffffffffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0001000000000000
header: deffffffffffff01000000000000

kid: 0x0000ffffffffffff
ctr: 0x00ffffffffffffff
header: deffffffffffffffffffffffffffff

kid: 0x0000ffffffffffff
ctr: 0x0100000000000000
header: dfffffffffffff0100000000000000

kid: 0x0000ffffffffffff
ctr: 0xffffffffffffffff
header: dfffffffffffffffffffffffffffffff

kid: 0x0001000000000000
ctr: 0x0000000000000000
header: e001000000000000
```

```
kid: 0x0001000000000000
ctr: 0x0000000000000001
header: e101000000000000

kid: 0x0001000000000000
ctr: 0x00000000000000ff
header: e801000000000000ff

kid: 0x0001000000000000
ctr: 0x0000000000000100
header: e9010000000000000100

kid: 0x0001000000000000
ctr: 0x000000000000ffff
header: e901000000000000ffff

kid: 0x0001000000000000
ctr: 0x0000000000010000
header: ea01000000000000010000

kid: 0x0001000000000000
ctr: 0x0000000000ffffff
header: ea01000000000000ffffff

kid: 0x0001000000000000
ctr: 0x0000000001000000
header: eb0100000000000001000000

kid: 0x0001000000000000
ctr: 0x00000000ffffffff
header: eb01000000000000ffffffff

kid: 0x0001000000000000
ctr: 0x0000000100000000
header: ec010000000000000100000000

kid: 0x0001000000000000
ctr: 0x000000ffffffffff
header: ec01000000000000ffffffffff

kid: 0x0001000000000000
ctr: 0x0000010000000000
header: ed01000000000000010000000000

kid: 0x0001000000000000
ctr: 0x0000ffffffffffff
header: ed01000000000000ffffffffffff
```

```
kid: 0x0001000000000000
ctr: 0x0001000000000000
header: ee0100000000000001000000000000

kid: 0x0001000000000000
ctr: 0x00ffffffffffffff
header: ee01000000000000ffffffffffffff

kid: 0x0001000000000000
ctr: 0x0100000000000000
header: ef010000000000000100000000000000

kid: 0x0001000000000000
ctr: 0xffffffffffffffff
header: ef01000000000000ffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000000000000000
header: e0ffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000000000000001
header: e1ffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x00000000000000ff
header: e8ffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000000000000100
header: e9ffffffffffffff0100

kid: 0x00ffffffffffffff
ctr: 0x000000000000ffff
header: e9ffffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000000000010000
header: eaffffffffffffff010000

kid: 0x00ffffffffffffff
ctr: 0x0000000000ffffff
header: eaffffffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000000001000000
header: ebffffffffffffff01000000
```

kid: 0x00ffffffffffffff
ctr: 0x00000000ffffffff
header: ebffffffffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000000100000000
header: ecffffffffffffff0100000000

kid: 0x00ffffffffffffff
ctr: 0x000000ffffffffff
header: ecffffffffffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0000010000000000
header: edffffffffffffff010000000000

kid: 0x00ffffffffffffff
ctr: 0x0000ffffffffffff
header: edffffffffffffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0001000000000000
header: eeffffffffffffff01000000000000

kid: 0x00ffffffffffffff
ctr: 0x00ffffffffffffff
header: eeffffffffffffffffffffffffffff

kid: 0x00ffffffffffffff
ctr: 0x0100000000000000
header: efffffffffffffff0100000000000000

kid: 0x00ffffffffffffff
ctr: 0xffffffffffffffff
header: efffffffffffffffffffffffffffffff

kid: 0x0100000000000000
ctr: 0x0000000000000000
header: f00100000000000000

kid: 0x0100000000000000
ctr: 0x0000000000000001
header: f10100000000000000

kid: 0x0100000000000000
ctr: 0x00000000000000ff
header: f80100000000000000ff

```
kid: 0x0100000000000000
ctr: 0x0000000000000100
header: f901000000000000000100

kid: 0x0100000000000000
ctr: 0x000000000000ffff
header: f90100000000000000ffff

kid: 0x0100000000000000
ctr: 0x0000000000010000
header: fa0100000000000000010000

kid: 0x0100000000000000
ctr: 0x0000000000ffffff
header: fa0100000000000000ffffff

kid: 0x0100000000000000
ctr: 0x0000000001000000
header: fb010000000000000001000000

kid: 0x0100000000000000
ctr: 0x00000000ffffffff
header: fb0100000000000000ffffffff

kid: 0x0100000000000000
ctr: 0x0000000100000000
header: fc01000000000000000100000000

kid: 0x0100000000000000
ctr: 0x000000ffffffffff
header: fc0100000000000000ffffffffff

kid: 0x0100000000000000
ctr: 0x0000010000000000
header: fd010000000000000001000000000000

kid: 0x0100000000000000
ctr: 0x0000ffffffffffff
header: fd0100000000000000ffffffffffff

kid: 0x0100000000000000
ctr: 0x0001000000000000
header: fe0100000000000000010000000000000000

kid: 0x0100000000000000
ctr: 0x00ffffffffffffff
header: fe0100000000000000ffffffffffffff
```

```
kid: 0x0100000000000000
ctr: 0x0100000000000000
header: ff010000000000000001000000000000
        00

kid: 0x0100000000000000
ctr: 0xffffffffffffffff
header: ff0100000000000000ffffffffffffff
        ff

kid: 0xffffffffffffffff
ctr: 0x0000000000000000
header: f0ffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0000000000000001
header: f1ffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x00000000000000ff
header: f8ffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0000000000000100
header: f9ffffffffffffffff0100

kid: 0xffffffffffffffff
ctr: 0x000000000000ffff
header: f9ffffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0000000000010000
header: faffffffffffffffff010000

kid: 0xffffffffffffffff
ctr: 0x0000000000ffffff
header: faffffffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0000000001000000
header: fbffffffffffffffff01000000

kid: 0xffffffffffffffff
ctr: 0x00000000ffffffff
header: fbffffffffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0000000100000000
header: fcffffffffffffffff0100000000
```

```
kid: 0xffffffffffffffff
ctr: 0x000000ffffffffff
header: fcffffffffffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0000010000000000
header: fdffffffffffffffff010000000000

kid: 0xffffffffffffffff
ctr: 0x0000ffffffffffff
header: fdffffffffffffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0001000000000000
header: feffffffffffffffff01000000000000

kid: 0xffffffffffffffff
ctr: 0x00ffffffffffffff
header: feffffffffffffffffffffffffffffff

kid: 0xffffffffffffffff
ctr: 0x0100000000000000
header: ffffffffffffffffff01000000000000
        00

kid: 0xffffffffffffffff
ctr: 0xffffffffffffffff
header: ffffffffffffffffffffffffffffffff
        ff
```

D.2. AEAD encryption/decryption using AES-CTR and HMAC

For each case, we provide:

- *cipher_suite: The index of the cipher suite in use (see Section 8.1)
- *key: The key input to encryption/decryption
- *enc_key: The encryption subkey produced by the derive_subkeys() algorithm
- *auth_key: The encryption subkey produced by the derive_subkeys() algorithm
- *nonce: The nonce input to encryption/decryption
- *aad: The aad input to encryption/decryption
- *pt: The plaintext

* ct: The ciphertext

An implementation should verify that the following are true, where AEAD.Encrypt and AEAD.Decrypt are as defined in Section 4.5.1:

* AEAD.Encrypt(key, nonce, aad, pt) == ct

* AEAD.Decrypt(key, nonce, aad, ct) == pt

The other values in the test vector are intermediate values provided to facilitate debugging of test failures.

```
cipher_suite: 0x0001
key: 000102030405060708090a0b0c0d0e0f
     101112131415161718191a1b1c1d1e1f
     202122232425262728292a2b2c2d2e2f
enc_key: 000102030405060708090a0b0c0d0e0f
auth_key: 101112131415161718191a1b1c1d1e1f
          202122232425262728292a2b2c2d2e2f
nonce: 101112131415161718191a1b
aad: 4945544620534672616d65205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 6339af04ada1d064688a442b8dc69d5b
    6bfa40f4bef0583e8081069cc60705

cipher_suite: 0x0002
key: 000102030405060708090a0b0c0d0e0f
     101112131415161718191a1b1c1d1e1f
     202122232425262728292a2b2c2d2e2f
enc_key: 000102030405060708090a0b0c0d0e0f
auth_key: 101112131415161718191a1b1c1d1e1f
          202122232425262728292a2b2c2d2e2f
nonce: 101112131415161718191a1b
aad: 4945544620534672616d65205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 6339af04ada1d064688a442b8dc69d5b
    6bfa40f4be6e93b7da076927bb
```

```
cipher_suite: 0x0003
key: 000102030405060708090a0b0c0d0e0f
     101112131415161718191a1b1c1d1e1f
     202122232425262728292a2b2c2d2e2f
enc_key: 000102030405060708090a0b0c0d0e0f
auth_key: 101112131415161718191a1b1c1d1e1f
          202122232425262728292a2b2c2d2e2f
nonce: 101112131415161718191a1b
aad: 4945544620534672616d65205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 6339af04ada1d064688a442b8dc69d5b
    6bfa40f4be09480509
```

D.3. SFrame encryption/decryption

For each case, we provide:

* cipher_suite: The index of the cipher suite in use (see Section 8.1)
* kid: A KID value
* ctr: A CTR value
* base_key: The base_key input to the derive_key_salt algorithm
* sframe_key_label: The label used to derive sframe_key in the derive_key_salt algorithm
* sframe_salt_label: The label used to derive sframe_salt in the derive_key_salt algorithm
* sframe_secret: The sframe_secret variable in the derive_key_salt algorithm
* sframe_key: The sframe_key value produced by the derive_key_salt algorithm
* sframe_salt: The sframe_salt value produced by the derive_key_salt algorithm
* metadata: The metadata input to the SFrame encrypt algorithm
* pt: The plaintext
* ct: The SFrame ciphertext

An implementation should verify that the following are true, where encrypt and decrypt are as defined in Section 4.4, using an SFrame context initialized with base_key assigned to kid:

*encrypt(ctr, kid, metadata, plaintext) == ct

*decrypt(metadata, ct) == pt

The other values in the test vector are intermediate values provided to facilitate debugging of test failures.

```
cipher_suite: 0x0001
kid: 0x0000000000000123
ctr: 0x0000000000004567
base_key: 000102030405060708090a0b0c0d0e0f
sframe_key_label: 534672616d6520312e30205365637265
                  74206b65792000000000000001230001
sframe_salt_label: 534672616d6520312e30205365637265
                   742073616c74200000000000000012300
                   01
sframe_secret: d926952ca8b7ec4a95941d1ada3a5203
               ceff8cceee34f574d23909eb314c40c0
sframe_key: 3f7d9a7c83ae8e1c8a11ae695ab59314
            b367e359fadac7b9c46b2bc6f81f46e1
            6b96f0811868d59402b7e870102720b3
sframe_salt: 50b29329a04dc0f184ac3168
metadata: 4945544620534672616d65205747
nonce: 50b29329a04dc0f184ac740f
aad: 99012345674945544620534672616d65
     205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 9901234567449408b6f490086165b9d6
    f62b24ae1a59a56486b4ae8ed036b889
    12e24f11
```

```
cipher_suite: 0x0002
kid: 0x0000000000000123
ctr: 0x0000000000004567
base_key: 000102030405060708090a0b0c0d0e0f
sframe_key_label: 534672616d6520312e30205365637265
                  74206b65792000000000000000001230002
sframe_salt_label: 534672616d6520312e30205365637265
                   742073616c74200000000000000012300
                   02
sframe_secret: d926952ca8b7ec4a95941d1ada3a5203
               ceff8cceee34f574d23909eb314c40c0
sframe_key: e2ec5c797540310483b16bf6e7a570d2
            a27d192fe869c7ccd8584a8d9dab9154
            9fbe553f5113461ec6aa83bf3865553e
sframe_salt: e68ac8dd3d02fbcd368c5577
metadata: 4945544620534672616d65205747
nonce: e68ac8dd3d02fbcd368c1010
aad: 99012345674945544620534672616d65
     205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 99012345673f31438db4d09434e43afa
    0f8a2f00867a2be085046a9f5cb4f101
    d607

cipher_suite: 0x0003
kid: 0x0000000000000123
ctr: 0x0000000000004567
base_key: 000102030405060708090a0b0c0d0e0f
sframe_key_label: 534672616d6520312e30205365637265
                  74206b65792000000000000000001230003
sframe_salt_label: 534672616d6520312e30205365637265
                   742073616c74200000000000000012300
                   03
sframe_secret: d926952ca8b7ec4a95941d1ada3a5203
               ceff8cceee34f574d23909eb314c40c0
sframe_key: 2c5703089cbb8c583475e4fc461d97d1
            8809df79b6d550f78eb6d50ffa80d892
            11d57909934f46f5405e38cd583c69fe
sframe_salt: 38c16e4f5159700c00c7f350
metadata: 4945544620534672616d65205747
nonce: 38c16e4f5159700c00c7b637
aad: 99012345674945544620534672616d65
     205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 990123456717fc8af28a5a695afcfc6c
    8df6358a17e26b2fcb3bae32e443
```

```
cipher_suite: 0x0004
kid: 0x0000000000000123
ctr: 0x0000000000004567
base_key: 000102030405060708090a0b0c0d0e0f
sframe_key_label: 534672616d6520312e30205365637265
                  74206b6579200000000000000001230004
sframe_salt_label: 534672616d6520312e30205365637265
                   742073616c742000000000000000012300
                   04
sframe_secret: d926952ca8b7ec4a95941d1ada3a5203
               ceff8cceee34f574d23909eb314c40c0
sframe_key: d34f547f4ca4f9a7447006fe7fcbf768
sframe_salt: 75234edefe07819026751816
metadata: 4945544620534672616d65205747
nonce: 75234edefe07819026755d71
aad: 99012345674945544620534672616d65
     205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 9901234567b7412c2513a1b66dbb4884
    1bbaf17f598751176ad847681a69c6d0
    b091c07018ce4adb34eb

cipher_suite: 0x0005
kid: 0x0000000000000123
ctr: 0x0000000000004567
base_key: 000102030405060708090a0b0c0d0e0f
sframe_key_label: 534672616d6520312e30205365637265
                  74206b6579200000000000000001230005
sframe_salt_label: 534672616d6520312e30205365637265
                   742073616c742000000000000000012300
                   05
sframe_secret: 0fc3ea6de6aac97a35f194cf9bed94d4
               b5230f1cb45a785c9fe5dce9c188938a
               b6ba005bc4c0a19181599e9d1bcf7b74
               aca48b60bf5e254e546d809313e083a3
sframe_key: d3e27b0d4a5ae9e55df01a70e6d4d28d
            969b246e2936f4b7a5d9b494da6b9633
sframe_salt: 84991c167b8cd23c93708ec7
metadata: 4945544620534672616d65205747
nonce: 84991c167b8cd23c9370cba0
aad: 99012345674945544620534672616d65
     205747
pt: 64726166742d696574662d736672616d
    652d656e63
ct: 990123456794f509d36e9beacb0e261d
    99c7d1e972f1fed787d4049f17ca2135
    3c1cc24d56ceabced279
```

Contributors

Frederic Jacobs
Apple

Email: frederic.jacobs@apple.com

Marta Mularczyk
Amazon

Email: mulmarta@amazon.com

Suhas Nandakumar
Cisco

Email: snandaku@cisco.com

Tomas Rigaux
Cisco

Email: trigaux@cisco.com

Raphael Robert
Phoenix R&D

Email: ietf@raphaelrobert.com

Authors' Addresses

Emad Omara
Apple

Email: eomara@apple.com

Justin Uberti
Google

Email: juberti@google.com

Sergio Garcia Murillo
CoSMo Software

Email: sergio.garcia.murillo@cosmosoftware.io

Richard L. Barnes (editor)
Cisco

Email: rlb@ipv.sx

Youenn Fablet
Apple

Email: youenn@apple.com