Internet Engineering Task Force (IETF)
Request for Comments: 6904
Updates: 3711
Category: Standards Track
ISSN: 2070-1721

J. Lennox
Vidyo

April 2013

Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)

Abstract

The Secure Real-time Transport Protocol (SRTP) provides authentication, but not encryption, of the headers of Real-time Transport Protocol (RTP) packets. However, RTP header extensions may carry sensitive information for which participants in multimedia sessions want confidentiality. This document provides a mechanism, extending the mechanisms of SRTP, to selectively encrypt RTP header extensions in SRTP.

This document updates RFC 3711, the Secure Real-time Transport Protocol specification, to require that all future SRTP encryption transforms specify how RTP header extensions are to be encrypted.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6904.

Table of Contents

1. Introduction

The Secure Real-time Transport Protocol [RFC3711] specification provides confidentiality, message authentication, and replay protection for multimedia payloads sent using the Real-time Protocol (RTP) [RFC3550]. However, in order to preserve RTP header compression efficiency, SRTP provides only authentication and replay protection for the headers of RTP packets, not confidentiality.

For the standard portions of an RTP header, providing only authentication and replay protection does not normally present a problem, as the information carried in an RTP header does not provide much information beyond that which an attacker could infer by observing the size and timing of RTP packets. Thus, there is little need for confidentiality of the header information.

However, the security requirements can be different for information carried in RTP header extensions. A number of recent proposals for header extensions using the mechanism described in "A General Mechanism for RTP Header Extensions" [RFC5285] carry information for which confidentiality could be desired or essential. Notably, two recent specifications ([RFC6464] and [RFC6465]) contain information about per-packet sound levels of the media data carried in the RTP payload and specify that exposing this information to an eavesdropper is unacceptable in many circumstances (as described in the Security Considerations sections of those RFCs).

This document, therefore, defines a mechanism by which encryption can be applied to RTP header extensions when they are transported using SRTP. As an RTP sender may wish some extension information to be sent in the clear (for example, it may be useful for a network monitoring device to be aware of RTP transmission time offsets [RFC5450]), this mechanism can be selectively applied to a subset of the header extension elements carried in an SRTP packet.

The mechanism defined by this document encrypts packets' header extensions using the same cryptographic algorithms and parameters as are used to encrypt the packets' RTP payloads. This document defines how this is done for the encryption transforms defined in [RFC3711], [RFC5669], and [RFC6188], which are the SRTP encryption transforms defined by Standards Track RFCs at the time of this writing. It also updates [RFC3711] to indicate that specifications of future SRTP encryption transforms must define how header extension encryption is to be performed.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

3. Encryption Mechanism

Encrypted header extension elements are carried in the same manner as non-encrypted header extension elements, as defined by [RFC5285]. The one- or two-byte header of the extension elements is not encrypted, nor is any of the header extension padding. If multiple different header extension elements are being encrypted, they have separate element identifier values, just as they would if they were not encrypted. Similarly, encrypted and non-encrypted header extension elements have separate identifier values.

Encrypted header extension elements are carried only in packets encrypted using the Secure Real-time Transport Protocol [RFC3711]. To encrypt (or decrypt) encrypted header extension elements, an SRTP participant first uses the SRTP key derivation algorithm, specified in Section 4.3.1 of [RFC3711], to generate header encryption and header salting keys, using the same pseudorandom function family as is used for the key derivation for the SRTP session. These keys are derived as follows:

- k_he (SRTP header encryption): <label> = 0x06, n=n_e.

- k_hs (SRTP header salting key): <label> = 0x07, n=n_s.

where n_e and n_s are from the cryptographic context: the same size encryption key and salting key are used as are used for the SRTP payload. Additionally, the same master key, master salt, index, and key_derivation_rate are used as for the SRTP payload. (Note that since RTP headers, including header extensions, are authenticated in SRTP, no new authentication key is needed for header extensions.)

A header extension keystream is generated for each packet containing encrypted header extension elements. The details of how this header extension keystream is generated depend on the encryption transform that is used for the SRTP packet. For encryption transforms that have been standardized as of the date of publication of this document, see Section 3.2; for requirements for new transforms, see Section 3.3.

After the header extension keystream is generated, the SRTP participant then computes an encryption mask for the header extension, identifying the portions of the header extension that are, or are to be, encrypted. (For an example of this procedure, see Section 3.1.) This encryption mask corresponds to the entire payload of each header extension element that is encrypted. It does not include any non-encrypted header extension elements, any extension element headers, or any padding octets. The encryption mask has all-bits-1 octets (i.e., hexadecimal 0xff) for header extension octets that are to be encrypted and all-bits-0 octets for header extension octets that are not to be encrypted. The set of extension elements to be encrypted is communicated between the sender and the receiver using the signaling mechanisms described in Section 4.

This encryption mask is computed separately for every packet that carries a header extension. Based on the non-encrypted portions of the headers and the signaled list of encrypted extension elements, a receiver can always determine the correct encryption mask for any encrypted header extension.

The SRTP participant bitwise-ANDs the encryption mask with the keystream to produce a masked keystream. It then bitwise exclusive-ORs the header extension with this masked keystream to produce the ciphertext version of the header extension. (Thus, octets indicated as all-bits-1 in the encrypted mask are encrypted, whereas those indicated as all-bits-0 are not.)

The header extension encryption process does not include the "defined by profile" or "length" fields of the header extension, only the field that Section 5.3.1 of [RFC3550] calls "header extension" proper, starting with the first [RFC5285] ID and length. Thus, both the encryption mask and the keystream begin at this point.

This header extension encryption process could, equivalently, be computed by considering the encryption mask as a mixture of the encrypted and unencrypted headers, i.e., as

```
EncryptedHeader = (Encrypt(Key, Plaintext) AND MASK) OR
                  (Plaintext AND (NOT MASK))
```

where Encrypt is the encryption function, MASK is the encryption mask, and AND, OR, and NOT are bitwise operations. This formulation of the encryption process might be preferred by implementations for which encryption is performed by a separate module and cannot be modified easily.

The SRTP authentication tag is computed across the encrypted header extension, i.e., the data that is actually transmitted on the wire. Thus, header extension encryption MUST be done before the authentication tag is computed, and authentication tag validation MUST be done on the encrypted header extensions. For receivers, header extension decryption SHOULD be done only after the receiver has validated the packet's message authentication tag, and the receiver MUST NOT take any actions based on decrypted headers, prior to validating the authentication tag, that could affect the security or proper functioning of the system.

3.1. Example Encryption Mask

If a sender wished to send a header extension containing an encrypted SMPTE timecode [RFC5484] with ID 1, a plaintext transmission time offset [RFC5450] with ID 2, an encrypted audio level indication [RFC6464] with ID 3, and an encrypted NTP timestamp [RFC6051] with ID 4, the plaintext RTP header extension might look like this:

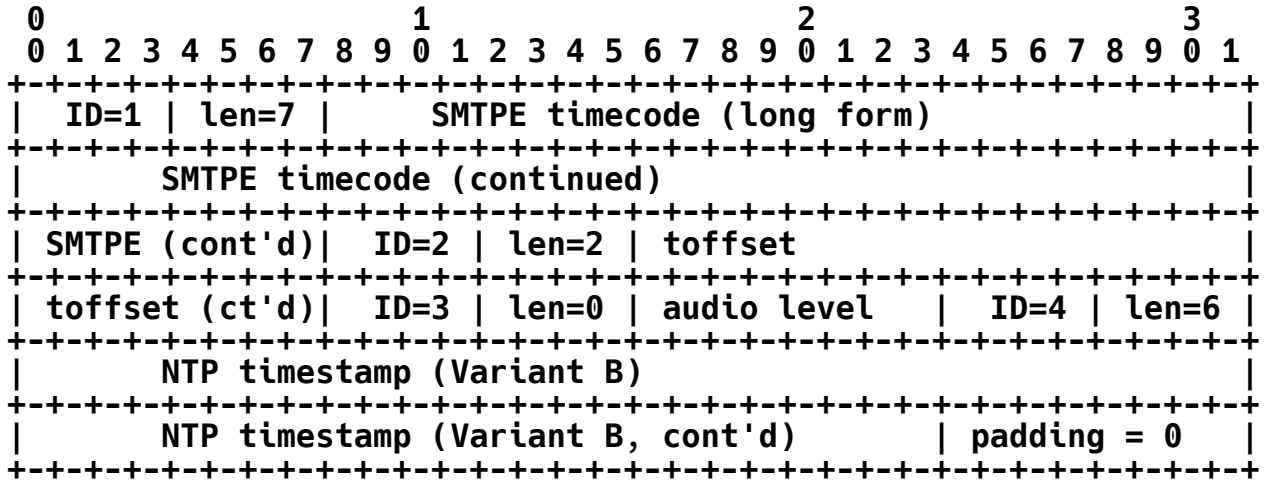

Figure 1: Structure of Plaintext Example Header Extension

The corresponding encryption mask would then be:

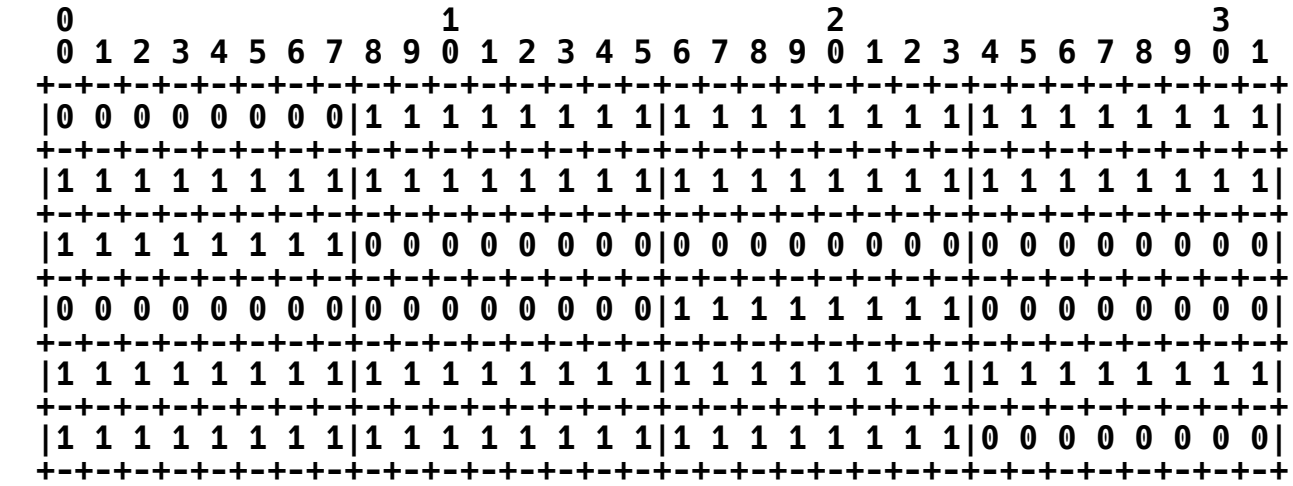

Figure 2: Encryption Mask for Example Header Extension

In the mask, the octets corresponding to the payloads of the encrypted header extension elements are set to all-1 values, and the octets corresponding to non-encrypted header extension elements, element headers, and header extension padding are set to all-zero values.

3.2. Header Extension Keystream Generation for Existing Encryption Transforms

For the AES-CM and AES-f8 transforms [RFC3711], the SEED-CTR transform [RFC5669], and the AES_192_CM and AES_256_CM transforms [RFC6188], the header extension keystream SHALL be generated for each packet containing encrypted header extension elements using the same encryption transform and Initialization Vector (IV) as are used for that packet's SRTP payload, except that the SRTP encryption and salting keys k_e and k_s are replaced by the SRTP header encryption and header salting keys k_he and k_hs, respectively, as defined above.

For the SEED-CCM and SEED-GCM transforms [RFC5669], the header extension keystream SHALL be generated using the algorithm specified above for the SEED-CTR algorithm. (Because the Authenticated Encryption with Associated Data (AEAD) transform used on the payload in these algorithms includes the RTP header, including the RTP header extension, in its Associated Authenticated Data (AAD), counter-mode encryption for the header extension is believed to be of equivalent cryptographic strength to the CCM and GCM transforms.)

For the NULL encryption transform [RFC3711], the header extension keystream SHALL be all-zero.

3.3. Header Extension Keystream Generation for Future Encryption Transforms

When new SRTP encryption transforms are defined, this document updates [RFC3711] as follows: in addition to the rules specified in Section 6 of RFC 3711, the Standards Track RFC defining the new transform MUST specify how the encryption transform is to be used with header extension encryption.

It is RECOMMENDED that new transformations follow the same mechanisms as are defined in Section 3.2 of this document if they are applicable and are believed to be cryptographically adequate for the transform in question.

4. Signaling (Setup) Information

Encrypted header extension elements are signaled in the Session Description Protocol (SDP) extmap attribute using the URI "urn:ietf:params:rtp-hdrext:encrypt" followed by the URI of the header extension element being encrypted, as well as any extensionattributes that extension normally takes. Figure 3 gives a formal Augmented Backus-Naur Form (ABNF) [RFC5234] showing this grammar extension, extending the grammar defined in [RFC5285].

```
enc-extensionname = %x75.72.6e.3a.69.65.74.66.3a.70.61.72.61.6d.73.3a
    %x72.74.70.2d.68.64.72.65.78.74.3a.65.6e.63.72.79.70.74
    ; "urn:ietf:params:rtp-hdrext:encrypt" in lower case

extmap =/ mapentry SP enc-extensionname SP extensionname
    [SP extensionattributes]

; extmap, mapentry, extensionname, and extensionattributes
; are defined in [RFC5285]
```

Figure 3: Syntax of the "encrypt" extmap

Thus, for example, to signal an SRTP session using encrypted SMPTE timecodes [RFC5484], while simultaneously signaling plaintext transmission time offsets [RFC5450], an SDP document could contain the text shown in Figure 4 (line breaks have been added for formatting).

```
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_32 \
  inline:NzB4d1BINUAvLEw6UzF3WSJ+PSdFcGdUJShpX1Zj|2^20|1:32
a=extmap:1 urn:ietf:params:rtp-hdrext:encrypt \
    urn:ietf:params:rtp-hdrext:smpte-tc 25@600/24
a=extmap:2 urn:ietf:params:rtp-hdrext:toffset
```

Figure 4: Sample SDP Document Offering Encrypted Headers

This example uses SDP security descriptions [RFC4568] for SRTP keying, but this is merely for illustration. Any SRTP keying mechanism to establish session keys will work.

The extmap SDP attribute is defined in [RFC5285] as being either a session or media attribute. If the extmap for an encrypted header extension is specified as a media attribute, it MUST be specified only for media that use SRTP-based RTP profiles. If such an extmap is specified as a session attribute, there MUST be at least one media in the SDP session that uses an SRTP-based RTP profile. The session-level extmap applies to all the SRTP-based media in the session and MUST be ignored for all other (non-SRTP or non-RTP) media.

The "urn:ietf:params:rtp-hdrext:encrypt" extension MUST NOT be recursively applied to itself.

4.1. Backward Compatibility

Following the procedures in [RFC5285], an SDP endpoint that does not understand the "urn:ietf:params:rtp-hdrext:encrypt" extension URI will ignore the extension and, for SDP offer/answer, will negotiate not to use it.

For backward compatibility with endpoints that do not implement this specification, in a negotiated session (whether using offer/answer or some other means), best-effort encryption of a header extension element is possible: an endpoint MAY offer the same header extension element both encrypted and unencrypted. An offerer MUST offer only best-effort negotiation when lack of confidentiality would be acceptable in the backward-compatible case. Answerers (or equivalent peers in a negotiation) that understand header extension encryption SHOULD choose the encrypted form of the offered header extension element and mark the unencrypted form "inactive", unless they have an explicit reason to prefer the unencrypted form. In all cases, answerers MUST NOT negotiate the use of, and senders MUST NOT send, both encrypted and unencrypted forms of the same header extension.

Note that, as always, users of best-effort encryption MUST be cautious of bid-down attacks, where a man-in-the-middle attacker removes a higher-security option, forcing endpoints to negotiate a lower-security one. Appropriate countermeasures depend on the signaling protocol in use, but users can ensure, for example, that signaling is integrity-protected.

5. Security Considerations

The security properties of header extension elements protected by the mechanism in this document are equivalent to those for SRTP payloads.

The mechanism defined in this document does not provide confidentiality about which header extension elements are used for a given SRTP packet, only for the content of those header extension elements. This appears to be in the spirit of SRTP itself, which does not encrypt RTP headers. If this is a concern, an alternate mechanism would be needed to provide confidentiality.

For the two-byte-header form of header extension elements (0x100N, where "N" is the appbits field), this mechanism does not provide any protection to zero-length header extension elements (for which their presence or absence is the only information they carry). It also does not provide any protection for the appbits (field 256, the lowest four bits of the "defined by profile" field) of the two-byte headers. Neither of these features is present in the one-byte-header form of header extension elements (0xBEDE), so these limitations do not apply in that case.

This mechanism cannot protect RTP header extensions that do not use the mechanism defined in [RFC5285].

This document does not specify the circumstances in which extension header encryption should be used. Documents defining specific header extension elements should provide guidance on when encryption is appropriate for these elements.

If a middlebox does not have access to the SRTP authentication keys, it has no way to verify the authenticity of unencrypted RTP header extension elements (or the unencrypted RTP header), even though it can monitor them. Therefore, such middleboxes MUST treat such headers as untrusted and potentially generated by an attacker, in the same way as they treat unauthenticated traffic. (This does not mean that middleboxes cannot view and interpret such traffic, of course, only that appropriate skepticism needs to be maintained about the results of such interpretation.)

There is no mechanism defined to protect header extensions with different algorithms or encryption keys than are used to protect the RTP payloads. In particular, it is not possible to provide confidentiality for a header extension while leaving the payload in cleartext.

The dangers of using weak or NULL authentication with SRTP, described in Section 9.5 of [RFC3711], apply to encrypted header extensions as well. In particular, since some header extension elements will have some easily guessed plaintext bits, strong authentication is REQUIRED if an attacker setting such bits could have a meaningful effect on the behavior of the system.

The technique defined in this document can be applied only to encryption transforms that work by generating a pseudorandom keystream and bitwise exclusive-ORing it with the plaintext, such as CTR or f8. It will not work with ECB, CBC, or any other encryption method that does not use a keystream.

6. IANA Considerations

This document defines a new extension URI to the RTP Compact Header Extensions subregistry of the Real-Time Transport Protocol (RTP) Parameters registry, according to the following data:

Extension URI:	urn:ietf:params:rtp-hdrext:encrypt
Description:	Encrypted header extension element
Contact:	jonathan@vidyo.com
Reference:	RFC 6904

7. Acknowledgments

Thanks to Benoit Claise, Roni Even, Stephen Farrell, Kevin Igoe, Joel Jaeggli, David McGrew, David Singer, Robert Sparks, Magnus Westerlund, Qin Wu, and Felix Wyss for their comments and suggestions in the development of this specification.

8. References

8.1. Normative References

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

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

[RFC5669] Yoon, S., Kim, J., Park, H., Jeong, H., and Y. Won, "The SEED Cipher Algorithm and Its Use with the Secure Real-Time Transport Protocol (SRTP)", RFC 5669, August 2010.

[RFC6188] McGrew, D., "The Use of AES-192 and AES-256 in Secure RTP", RFC 6188, March 2011.

8.2. Informative References

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

[RFC5450] Singer, D. and H. Desineni, "Transmission Time Offsets in RTP Streams", RFC 5450, March 2009.

[RFC5484] Singer, D., "Associating Time-Codes with RTP Streams", RFC 5484, March 2009.

[RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.

[RFC6464] Lennox, J., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.

[RFC6465] Ivov, E., Marocco, E., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, December 2011.

Appendix A. Test Vectors

A.1. Key Derivation Test Vectors

This section provides test data for the header extension key derivation function, using AES-128 in Counter Mode. (The algorithms and keys used are the same as those for the test vectors in Appendix B.3 of [RFC3711].)

The inputs to the key derivation function are the 16-octet master key and the 14-octet master salt:

master key: E1F97A0D3E018BE0D64FA32C06DE4139

master salt: 0EC675AD498AFEEBB6960B3AABE6

Following [RFC3711], the input block for AES-CM is generated by exclusive-ORing the master salt with the concatenation of the encryption key label 0x06 with (index DIV kdr), then padding on the right with two null octets, which implements the multiply-by-2^16 operation (see Section 4.3.3 of [RFC3711]). The resulting value is then AES-CM-encrypted using the master key to get the cipher key.

```
index DIV kdr:                    000000000000
label:                          06
master salt:      0EC675AD498AFEEBB6960B3AABE6
-------------------------------------------------
XOR:              0EC675AD498AFEEDB6960B3AABE6     (x, PRF input)

x*2^16:           0EC675AD498AFEEDB6960B3AABE60000 (AES-CM input)

hdr. cipher key:  549752054D6FB708622C4A2E596A1B93 (AES-CM output)
```

Next, we show how the cipher salt is generated. The input block for AES-CM is generated by exclusive-ORing the master salt with the concatenation of the encryption salt label. That value is padded and encrypted as above.

```
index DIV kdr:                    000000000000
label:                         07
master salt:      0EC675AD498AFEEBB6960B3AABE6

------------------------------------------------
XOR:              0EC675AD498AFEECB6960B3AABE6     (x, PRF input)

x*2^16:           0EC675AD498AFEECB6960B3AABE60000 (AES-CM input)

                  AB01818174C40D39A3781F7C2D270733 (AES-CM ouptut)

hdr. cipher salt: AB01818174C40D39A3781F7C2D27
```

A.2. Header Encryption Test Vectors Using AES-CM

This section provides test vectors for the encryption of a header extension using the AES_CM cryptographic transform.

The header extension is encrypted using the header cipher key and header cipher salt computed in Appendix A.1. The header extension is carried in an SRTP-encrypted RTP packet with SSRC 0xCAFEBABE, sequence number 0x1234, and an all-zero rollover counter.

```
Session Key:      549752054D6FB708622C4A2E596A1B93
Session Salt:     AB01818174C40D39A3781F7C2D27

SSRC:                       CAFEBABE
Rollover Counter:                  00000000
Sequence Number:                         1234
------------------------------------------------
Init. Counter:    AB018181BE3AB787A3781F7C3F130000
```

The SRTP session was negotiated to indicate that header extension ID values 1, 3, and 4 are encrypted.

In hexadecimal, the header extension being encrypted is as follows (spaces have been added to show the internal structure of the header extension):

```
17 414273A475262748 22 0000C8 30 8E 46 55996386B395FB 00
```

This header extension is 24 bytes long. (Its values are intended to represent plausible values of the header extension elements shown in Section 3.1, but their specific meaning is not important for the example.) The header extension "defined by profile" and "length" fields, which in this case are BEDE 0006 in hexadecimal, are not included in the encryption process.

In hexadecimal, the corresponding encryption mask selecting the bodies of header extensions 1, 2, and 4 (corresponding to the mask in Figure 2) is:

```
   00 FFFFFFFFFFFFFFFF 00 000000 00 FF 00 FFFFFFFFFFFFFF 00
```

Finally, we compute the keystream from the session key and the initial counter, apply the mask to the keystream, and then exclusive-OR the keystream with the plaintext:

```
    Initial keystream:  1E19C8E1D481C779549ED1617AAA1B7A
                        FC0D933AE7ED6CC8
    Mask (hex):         00FFFFFFFFFFFFFFFF0000000000FF00
                        FFFFFFFFFFFFFF00
    Masked keystream:   0019C8E1D481C7795400000000001B00
                        FC0D933AE7ED6C00
    Plaintext:          17414273A4752627482200000C8308E46
                        55996386B395FB00
    Ciphertext:         17588A9270F4E15E1C220000C8309546
                        A994F0BC54789700
```

Author's Address

Jonathan Lennox
Vidyo, Inc.
433 Hackensack Avenue
Seventh Floor
Hackensack, NJ 07601
US

EMail: jonathan@vidyo.com