Network Working Group
Internet Draft
Intended Status: Informational
Expires: July 28, 2016

M. Jenkins
National Security Agency
M. Peck
The MITRE Corporation
K. Burgin

January 25, 2016

AES Encryption with HMAC-SHA2 for Kerberos 5
draft-ietf-kitten-aes-cts-hmac-sha2-09

Abstract

This document specifies two encryption types and two corresponding checksum types for Kerberos 5. The new types use AES in CTS mode (CBC mode with ciphertext stealing) for confidentiality and HMAC with a SHA-2 hash for integrity.

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 http://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 July 28, 2016.

Copyright and License Notice

Copyright (c) 2016 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 (http://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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Protocol Key Representation 3
3. Key Derivation Function 3
4. Key Generation from Pass Phrases 4
5. Kerberos Algorithm Protocol Parameters 5
6. Checksum Parameters 7
7. IANA Considerations 7
8. Security Considerations 8
8.1. Random Values in Salt Strings 8
8.2. Algorithm Rationale 9
9. Acknowledgements 9
10. References 9
10.1. Normative References 9
10.2. Informative References 9
Appendix A. Test Vectors 10
Authors' Addresses 17

1. Introduction

This document defines two encryption types and two corresponding checksum types for Kerberos 5 using AES with 128-bit or 256-bit keys.

To avoid ciphertext expansion, we use a variation of the CBC-CS3 mode defined in [SP800-38A+], also referred to as ciphertext stealing or CTS mode. The new types conform to the framework specified in [RFC3961], but do not use the simplified profile.

The encryption and checksum types defined in this document are intended to support environments that desire to use SHA-256 or SHA-384 as the hash algorithm. Differences between the encryption and checksum types defined in this document and the pre-existing Kerberos AES encryption and checksum types specified in [RFC3962] are:

* The pseudorandom function used by PBKDF2 is HMAC-SHA-256 or HMAC-SHA-384.

* A key derivation function from [SP800-108] using the SHA-256 or SHA-384 hash algorithm is used to produce keys for encryption, integrity protection, and checksum operations.

* The HMAC is calculated over the cipherstate concatenated with the AES output, instead of being calculated over the confounder and plaintext. This allows the message receiver to verify the integrity of the message before decrypting the message.

* The HMAC algorithm uses the SHA-256 or SHA-384 hash algorithm for integrity protection and checksum operations.

2. Protocol Key Representation

The AES key space is dense, so we can use random or pseudorandom octet strings directly as keys. The byte representation for the key is described in [FIPS197], where the first bit of the bit string is the high bit of the first byte of the byte string (octet string).

3. Key Derivation Function

We use a key derivation function from Section 5.1 of [SP800-108] which uses the HMAC algorithm as the PRF.

```
KDF-HMAC-SHA2(key, label, k) = k-truncate(K1)
```

key: The source of entropy from which subsequent keys are derived (this is known as Ki in [SP800-108]).

label: An octet string describing the intended usage of the derived key.

k: Length in bits of the key to be outputted, expressed in big-endian binary representation in 4 bytes (this is known as L in [SP800-108]).
(e.g. k = 128 is represented as 0x00000080,
k = 192 as 0x000000C0, k = 256 as 0x00000100,
k = 384 as 0x00000180)

When the encryption type is aes128-cts-hmac-sha256-128, k must be no greater than 256. When the encryption type is aes256-cts-hmac-sha384-192, k must be no greater than 384.

The k-truncate function is defined in [RFC3961], Section 5.1.

In all computations in this document, | indicates concatenation.

When the encryption type is aes128-cts-hmac-sha256-128, then K1 is computed as follows:

```
K1 = HMAC-SHA-256(key, 0x00000001 | label | 0x00 | k)
```

When the encryption type is aes256-cts-hmac-sha384-192, then K1 is computed as follows:

```
K1 = HMAC-SHA-384(key, 0x00000001 | label | 0x00 | k)
```

4. Key Generation from Pass Phrases

As defined below, the string-to-key function uses PBKDF2 [RFC2898] and KDF-HMAC-SHA2 to derive the base-key from a passphrase and salt. The string-to-key parameter string is four octets indicating an unsigned number in big-endian order, consistent with [RFC3962], except that the default is decimal 32768 if the parameter is not specified.

To ensure that different long-term base-keys are used with different enctypes, we prepend the enctype name to the salt, separated by a null byte. The enctype-name is "aes128-cts-hmac-sha256-128" or "aes256-cts-hmac-sha384-192" (without the quotes).

The user's long-term base-key is derived as follows:

```
iter_count = string-to-key parameter, default is decimal 32768
saltp = enctype-name | 0x00 | salt
tkey = random-to-key(PBKDF2(passphrase, saltp,
                            iter_count, keylength))
base-key = random-to-key(KDF-HMAC-SHA2(tkey, "kerberos",
```

```
                                        keylength))
                where "kerberos" is the octet-string
                0x6B65726265726F73
```

where the pseudorandom function used by PBKDF2 is HMAC-SHA-256 when the enctype is "aes128-cts-hmac-sha256-128" and HMAC-SHA-384 when the enctype is "aes256-cts-hmac-sha384-192", the value for keylength is the AES key length (128 or 256 bits), and the algorithm KDF-HMAC-SHA2 is defined in Section 3.

5. Kerberos Algorithm Protocol Parameters

The cipherstate is used as the formal initialization vector (IV) input into CBC-CS3. The plaintext is prepended with a 16-octet random nonce generated by the message originator, known as a confounder.

The ciphertext is a concatenation of the output of AES in CBC-CS3 mode and the HMAC of the cipherstate concatenated with the AES output. The HMAC is computed using either SHA-256 or SHA-384 depending on the encryption type. The output of HMAC-SHA-256 is truncated to 128 bits and the output of HMAC-SHA-384 is truncated to 192 bits. Sample test vectors are given in Appendix A.

Decryption is performed by removing the HMAC, verifying the HMAC against the cipherstate concatenated with the ciphertext, and then decrypting the ciphertext if the HMAC is correct. Finally, the first 16 octets of the decryption output (the confounder) is discarded, and the remainder is returned as the plaintext decryption output.

The following parameters apply to the encryption types aes128-cts-hmac-sha256-128 and aes256-cts-hmac-sha384-192.

protocol key format: as defined in Section 2.

specific key structure: three derived keys: { Kc, Ke, Ki }.

Kc: the checksum key, inputted into HMAC to provide the checksum mechanism defined in Section 6.

Ke: the encryption key, inputted into AES encryption and decryption as defined in "encryption function" and "decryption function" below.

Ki: the integrity key, inputted into HMAC to provide authenticated encryption as defined in "encryption function" and "decryption function" below.

required checksum mechanism: as defined in Section 6.

key-generation seed length: key size (128 or 256 bits).

string-to-key function: as defined in Section 4.

default string-to-key parameters: decimal 32768.

random-to-key function: identity function.

key-derivation function: KDF-HMAC-SHA2 as defined in Section 3. The key usage number is expressed as four octets in big-endian order.

```
If the enctype is aes128-cts-hmac-sha256-128:
Kc = KDF-HMAC-SHA2(base-key, usage | 0x99, 128)
Ke = KDF-HMAC-SHA2(base-key, usage | 0xAA, 128)
Ki = KDF-HMAC-SHA2(base-key, usage | 0x55, 128)

If the enctype is aes256-cts-hmac-sha384-192:
Kc = KDF-HMAC-SHA2(base-key, usage | 0x99, 192)
Ke = KDF-HMAC-SHA2(base-key, usage | 0xAA, 256)
Ki = KDF-HMAC-SHA2(base-key, usage | 0x55, 192)
```

cipherstate: a 128-bit CBC initialization vector derived from the ciphertext.

initial cipherstate: all bits zero.

encryption function: as follows, where E() is AES encryption in CBC-CS3 mode, and h is the size of truncated HMAC (128 bits or 192 bits as described above).

```
N = random nonce of length 128 bits (the AES block size)
IV = cipherstate
C = E(Ke, N | plaintext, IV)
H = HMAC(Ki, IV | C)
ciphertext = C | H[1..h]

Steps to compute the 128-bit cipherstate:
 L = length of C in bits
 portion C into 128-bit blocks, placing any remainder
 of less than 128 bits into a final block
 if L == 128: cipherstate = C
 else if L mod 128 > 0: cipherstate = last full (128-bit)
                                      block of C (the
                                      next-to-last block)
 else if L mod 128 == 0: cipherstate = next-to-last block
                                       of C
```

decryption function: as follows, where D() is AES decryption in

CBC-CS3 mode, and h is the size of truncated HMAC.

```
(C, H) = ciphertext
IV = cipherstate
if H != HMAC(Ki, IV | C)[1..h]
    stop, report error
(N, P) = D(Ke, C, IV)
Note: N is set to the first block of the decryption output,
P is set to the rest of the output.
cipherstate = same as described above in encryption function
```

pseudo-random function:

```
If the enctype is aes128-cts-hmac-sha256-128:
PRF = KDF-HMAC-SHA2(base-key, "prf" | octet-string, 256)

If the enctype is aes256-cts-hmac-sha384-192:
PRF = KDF-HMAC-SHA2(base-key, "prf" | octet-string, 384)

where "prf" is the octet-string 0x707266
```

6. Checksum Parameters

The following parameters apply to the checksum types hmac-sha256-128-aes128 and hmac-sha384-192-aes256, which are the associated checksums for aes128-cts-hmac-sha256-128 and aes256-cts-hmac-sha384-192, respectively.

associated cryptosystem: AES-128-CTS or AES-256-CTS as appropriate.

```
get_mic: HMAC(Kc, message)[1..h].
    where h is 128 bits for checksum type hmac-sha256-128-aes128
    and 192 bits for checksum type hmac-sha384-192-aes256

verify_mic: get_mic and compare.
```

7. IANA Considerations

IANA is requested to assign:

Encryption type numbers for aes128-cts-hmac-sha256-128 and aes256-cts-hmac-sha384-192 in the Kerberos Encryption Type Numbers registry.

Etype	Encryption type	Reference
TBD1	aes128-cts-hmac-sha256-128	[this document]

TBD2	aes256-cts-hmac-sha384-192	[this document]

Checksum type numbers for hmac-sha256-128-aes128 and hmac-sha384-192-aes256 in the Kerberos Checksum Type Numbers registry.

Sumtype	Checksum type	Size	Reference
TBD3	hmac-sha256-128-aes128	16	[this document]
TBD4	hmac-sha384-192-aes256	24	[this document]

8. Security Considerations

This specification requires implementations to generate random values. The use of inadequate pseudo-random number generators (PRNGs) can result in little or no security. The generation of quality random numbers is difficult. [RFC4086] offers random number generation guidance.

This document specifies a mechanism for generating keys from passphrases or passwords. The use of PBKDF2, a salt, and a large iteration count adds some resistance to off-line dictionary attacks by passive eavesdroppers. Salting prevents rainbow table attacks, while large iteration counts slow password guess attempts. Nonetheless, it is important to choose strong passphrases. Use of other Kerberos extensions that protect against off-line dictionary attacks should also be considered.

The NIST guidance in section 5.3 of [SP800-38A], requiring that CBC initialization vectors be unpredictable, is satisfied by the use of a random confounder as the first block of plaintext. The confounder fills the cryptographic role typically played by an initialization vector. This approach was chosen to align with other Kerberos cryptosystem approaches.

8.1. Random Values in Salt Strings

NIST guidance in Section 5.1 of [SP800-132] requires at least 128 bits of the salt to be randomly generated. The string-to-key function as defined in [RFC3961] requires the salt to be valid UTF-8 strings. Not every 128-bit random string will be valid UTF-8, so a UTF-8 compatible encoding would be needed to encapsulate the random bits. However, using a salt containing a random portion may have the following issues with some implementations:

* Cross-realm TGTs are typically managed by entering the same password at two KDCs to get the same keys. If each KDC uses a random salt, they won't have the same keys.

* Random salts may interfere with password history checking.

8.2. Algorithm Rationale

This document has been written to be consistent with common implementations of AES and SHA-2. The encryption and hash algorithm sizes have been chosen to create a consistent level of protection, with consideration to implementation efficiencies. So, for instance, SHA-384, which would normally be matched to AES-192, is instead matched to AES-256 to leverage the fact that there are efficient hardware implementations of AES-256. Note that, as indicated by the enc-type name "aes256-cts-hmac-sha384-192", the truncation of the HMAC-SHA-384 output to 192-bits results in an overall 192-bit level of security.

9. Acknowledgements

Kelley Burgin was employed at the National Security Agency during much of the work on this document.

10. References

10.1. Normative References

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[FIPS197] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, November 2001.

[SP800-38A+] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode", NIST Special Publication 800-38A Addendum, October 2010.

[SP800-108] National Institute of Standards and Technology, "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, October 2009.

10.2. Informative References

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[SP800-38A] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", NIST Special Publication 800-38A, December 2001.

[SP800-132] National Institute of Standards and Technology, "Recommendation for Password-Based Key Derivation, Part 1: Storage Applications", NIST Special Publication 800-132, June 2010.

Appendix A. Test Vectors

```
Sample results for string-to-key conversion:
--------------------------------------------

Iteration count = 32768
Pass phrase = "password"
Saltp for creating 128-bit base-key:
   61 65 73 31 32 38 2D 63 74 73 2D 68 6D 61 63 2D
   73 68 61 32 35 36 2D 31 32 38 00 10 DF 9D D7 83
   E5 BC 8A CE A1 73 0E 74 35 5F 61 41 54 48 45 4E
   41 2E 4D 49 54 2E 45 44 55 72 61 65 62 75 72 6E

(The saltp is "aes128-cts-hmac-sha256-128" | 0x00 |
 random 16 byte valid UTF-8 sequence | "ATHENA.MIT.EDUraeburn")
128-bit base-key:
   08 9B CA 48 B1 05 EA 6E A7 7C A5 D2 F3 9D C5 E7

Saltp for creating 256-bit base-key:
   61 65 73 32 35 36 2D 63 74 73 2D 68 6D 61 63 2D
   73 68 61 33 38 34 2D 31 39 32 00 10 DF 9D D7 83
   E5 BC 8A CE A1 73 0E 74 35 5F 61 41 54 48 45 4E
   41 2E 4D 49 54 2E 45 44 55 72 61 65 62 75 72 6E
(The saltp is "aes256-cts-hmac-sha384-192" | 0x00 |
 random 16 byte valid UTF-8 sequence | "ATHENA.MIT.EDUraeburn")
256-bit base-key:
   45 BD 80 6D BF 6A 83 3A 9C FF C1 C9 45 89 A2 22
   36 7A 79 BC 21 C4 13 71 89 06 E9 F5 78 A7 84 67

Sample results for key derivation:
----------------------------------

enctype aes128-cts-hmac-sha256-128:
128-bit base-key:
```

```
      37 05 D9 60 80 C1 77 28 A0 E8 00 EA B6 E0 D2 3C
   Kc value for key usage 2 (constant = 0x0000000299):
      B3 1A 01 8A 48 F5 47 76 F4 03 E9 A3 96 32 5D C3
   Ke value for key usage 2 (constant = 0x00000002AA):
      9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
   Ki value for key usage 2 (constant = 0x0000000255):
      9F DA 0E 56 AB 2D 85 E1 56 9A 68 86 96 C2 6A 6C

   enctype aes256-cts-hmac-sha384-192:
   256-bit base-key:
      6D 40 4D 37 FA F7 9F 9D F0 D3 35 68 D3 20 66 98
      00 EB 48 36 47 2E A8 A0 26 D1 6B 71 82 46 0C 52
   Kc value for key usage 2 (constant = 0x0000000299):
      EF 57 18 BE 86 CC 84 96 3D 8B BB 50 31 E9 F5 C4
      BA 41 F2 8F AF 69 E7 3D
   Ke value for key usage 2 (constant = 0x00000002AA):
      56 AB 22 BE E6 3D 82 D7 BC 52 27 F6 77 3F 8E A7
      A5 EB 1C 82 51 60 C3 83 12 98 0C 44 2E 5C 7E 49
   Ki value for key usage 2 (constant = 0x0000000255):
      69 B1 65 14 E3 CD 8E 56 B8 20 10 D5 C7 30 12 B6
      22 C4 D0 0F FC 23 ED 1F

   Sample encryptions (all using the default cipher state):
   --------------------------------------------------------
   These sample encryptions use the above sample key
   derivation results, including use of the same
   base-key and key usage values.

   The following test vectors are for
   enctype aes128-cts-hmac-sha256-128:

   Plaintext: (empty)
   Confounder:
      7E 58 95 EA F2 67 24 35 BA D8 17 F5 45 A3 71 48
   128-bit AES key (Ke):
      9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
   128-bit HMAC key (Ki):
      9F DA 0E 56 AB 2D 85 E1 56 9A 68 86 96 C2 6A 6C
   AES Output:
      EF 85 FB 89 0B B8 47 2F 4D AB 20 39 4D CA 78 1D
   Truncated HMAC Output:
      AD 87 7E DA 39 D5 0C 87 0C 0D 5A 0A 8E 48 C7 18
   Ciphertext (AES Output | HMAC Output):
      EF 85 FB 89 0B B8 47 2F 4D AB 20 39 4D CA 78 1D
      AD 87 7E DA 39 D5 0C 87 0C 0D 5A 0A 8E 48 C7 18

   Plaintext: (length less than block size)
      00 01 02 03 04 05
```

```
Confounder:
   7B CA 28 5E 2F D4 13 0F B5 5B 1A 5C 83 BC 5B 24
128-bit AES key (Ke):
   9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
128-bit HMAC key (Ki):
   9F DA 0E 56 AB 2D 85 E1 56 9A 68 86 96 C2 6A 6C
AES Output:
   84 D7 F3 07 54 ED 98 7B AB 0B F3 50 6B EB 09 CF
   B5 54 02 CE F7 E6
Truncated HMAC Output:
   87 7C E9 9E 24 7E 52 D1 6E D4 42 1D FD F8 97 6C
Ciphertext:
   84 D7 F3 07 54 ED 98 7B AB 0B F3 50 6B EB 09 CF
   B5 54 02 CE F7 E6 87 7C E9 9E 24 7E 52 D1 6E D4
   42 1D FD F8 97 6C

Plaintext: (length equals block size)
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Confounder:
   56 AB 21 71 3F F6 2C 0A 14 57 20 0F 6F A9 94 8F
128-bit AES key (Ke):
   9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
128-bit HMAC key (Ki):
   9F DA 0E 56 AB 2D 85 E1 56 9A 68 86 96 C2 6A 6C
AES Output:
   35 17 D6 40 F5 0D DC 8A D3 62 87 22 B3 56 9D 2A
   E0 74 93 FA 82 63 25 40 80 EA 65 C1 00 8E 8F C2
Truncated HMAC Output:
   95 FB 48 52 E7 D8 3E 1E 7C 48 C3 7E EB E6 B0 D3
Ciphertext:
   35 17 D6 40 F5 0D DC 8A D3 62 87 22 B3 56 9D 2A
   E0 74 93 FA 82 63 25 40 80 EA 65 C1 00 8E 8F C2
   95 FB 48 52 E7 D8 3E 1E 7C 48 C3 7E EB E6 B0 D3

Plaintext: (length greater than block size)
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14
Confounder:
   A7 A4 E2 9A 47 28 CE 10 66 4F B6 4E 49 AD 3F AC
128-bit AES key (Ke):
   9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
128-bit HMAC key (Ki):
   9F DA 0E 56 AB 2D 85 E1 56 9A 68 86 96 C2 6A 6C
AES Output:
   72 0F 73 B1 8D 98 59 CD 6C CB 43 46 11 5C D3 36
   C7 0F 58 ED C0 C4 43 7C 55 73 54 4C 31 C8 13 BC
   E1 E6 D0 72 C1
Truncated HMAC Output:
```

```
      86 B3 9A 41 3C 2F 92 CA 9B 83 34 A2 87 FF CB FC
   Ciphertext:
      72 0F 73 B1 8D 98 59 CD 6C CB 43 46 11 5C D3 36
      C7 0F 58 ED C0 C4 43 7C 55 73 54 4C 31 C8 13 BC
      E1 E6 D0 72 C1 86 B3 9A 41 3C 2F 92 CA 9B 83 34
      A2 87 FF CB FC
```

The following test vectors are for enctype aes256-cts-hmac-sha384-192:

```
   Plaintext: (empty)
   Confounder:
      F7 64 E9 FA 15 C2 76 47 8B 2C 7D 0C 4E 5F 58 E4
   256-bit AES key (Ke):
      56 AB 22 BE E6 3D 82 D7 BC 52 27 F6 77 3F 8E A7
      A5 EB 1C 82 51 60 C3 83 12 98 0C 44 2E 5C 7E 49
   192-bit HMAC key (Ki):
      69 B1 65 14 E3 CD 8E 56 B8 20 10 D5 C7 30 12 B6
      22 C4 D0 0F FC 23 ED 1F
   AES Output:
      41 F5 3F A5 BF E7 02 6D 91 FA F9 BE 95 91 95 A0
   Truncated HMAC Output:
      58 70 72 73 A9 6A 40 F0 A0 19 60 62 1A C6 12 74
      8B 9B BF BE 7E B4 CE 3C
   Ciphertext:
      41 F5 3F A5 BF E7 02 6D 91 FA F9 BE 95 91 95 A0
      58 70 72 73 A9 6A 40 F0 A0 19 60 62 1A C6 12 74
      8B 9B BF BE 7E B4 CE 3C

   Plaintext: (length less than block size)
      00 01 02 03 04 05
   Confounder:
      B8 0D 32 51 C1 F6 47 14 94 25 6F FE 71 2D 0B 9A
   256-bit AES key (Ke):
      56 AB 22 BE E6 3D 82 D7 BC 52 27 F6 77 3F 8E A7
      A5 EB 1C 82 51 60 C3 83 12 98 0C 44 2E 5C 7E 49
   192-bit HMAC key (Ki):
      69 B1 65 14 E3 CD 8E 56 B8 20 10 D5 C7 30 12 B6
      22 C4 D0 0F FC 23 ED 1F
   AES Output:
      4E D7 B3 7C 2B CA C8 F7 4F 23 C1 CF 07 E6 2B C7
      B7 5F B3 F6 37 B9
   Truncated HMAC Output:
      F5 59 C7 F6 64 F6 9E AB 7B 60 92 23 75 26 EA 0D
      1F 61 CB 20 D6 9D 10 F2
   Ciphertext:
      4E D7 B3 7C 2B CA C8 F7 4F 23 C1 CF 07 E6 2B C7
      B7 5F B3 F6 37 B9 F5 59 C7 F6 64 F6 9E AB 7B 60
```

```
      92 23 75 26 EA 0D 1F 61 CB 20 D6 9D 10 F2

   Plaintext: (length equals block size)
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   Confounder:
      53 BF 8A 0D 10 52 65 D4 E2 76 42 86 24 CE 5E 63
   256-bit AES key (Ke):
      56 AB 22 BE E6 3D 82 D7 BC 52 27 F6 77 3F 8E A7
      A5 EB 1C 82 51 60 C3 83 12 98 0C 44 2E 5C 7E 49
   192-bit HMAC key (Ki):
      69 B1 65 14 E3 CD 8E 56 B8 20 10 D5 C7 30 12 B6
      22 C4 D0 0F FC 23 ED 1F
   AES Output:
      BC 47 FF EC 79 98 EB 91 E8 11 5C F8 D1 9D AC 4B
      BB E2 E1 63 E8 7D D3 7F 49 BE CA 92 02 77 64 F6
   Truncated HMAC Output:
      8C F5 1F 14 D7 98 C2 27 3F 35 DF 57 4D 1F 93 2E
      40 C4 FF 25 5B 36 A2 66
   Ciphertext:
      BC 47 FF EC 79 98 EB 91 E8 11 5C F8 D1 9D AC 4B
      BB E2 E1 63 E8 7D D3 7F 49 BE CA 92 02 77 64 F6
      8C F5 1F 14 D7 98 C2 27 3F 35 DF 57 4D 1F 93 2E
      40 C4 FF 25 5B 36 A2 66

   Plaintext: (length greater than block size)
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      10 11 12 13 14
   Confounder:
      76 3E 65 36 7E 86 4F 02 F5 51 53 C7 E3 B5 8A F1
   256-bit AES key (Ke):
      56 AB 22 BE E6 3D 82 D7 BC 52 27 F6 77 3F 8E A7
      A5 EB 1C 82 51 60 C3 83 12 98 0C 44 2E 5C 7E 49
   192-bit HMAC key (Ki):
      69 B1 65 14 E3 CD 8E 56 B8 20 10 D5 C7 30 12 B6
      22 C4 D0 0F FC 23 ED 1F
   AES Output:
      40 01 3E 2D F5 8E 87 51 95 7D 28 78 BC D2 D6 FE
      10 1C CF D5 56 CB 1E AE 79 DB 3C 3E E8 64 29 F2
      B2 A6 02 AC 86
   Truncated HMAC Output:
      FE F6 EC B6 47 D6 29 5F AE 07 7A 1F EB 51 75 08
      D2 C1 6B 41 92 E0 1F 62
   Ciphertext:
      40 01 3E 2D F5 8E 87 51 95 7D 28 78 BC D2 D6 FE
      10 1C CF D5 56 CB 1E AE 79 DB 3C 3E E8 64 29 F2
      B2 A6 02 AC 86 FE F6 EC B6 47 D6 29 5F AE 07 7A
      1F EB 51 75 08 D2 C1 6B 41 92 E0 1F 62
```

```
Sample checksums:
-----------------
These sample checksums use the above sample key
derivation results, including use of the same
base-key and key usage values.

Checksum type: hmac-sha256-128-aes128
128-bit HMAC key (Kc):
   B3 1A 01 8A 48 F5 47 76 F4 03 E9 A3 96 32 5D C3
Plaintext:
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14
Checksum:
   D7 83 67 18 66 43 D6 7B 41 1C BA 91 39 FC 1D EE

Checksum type: hmac-sha384-192-aes256
192-bit HMAC key (Kc):
   EF 57 18 BE 86 CC 84 96 3D 8B BB 50 31 E9 F5 C4
   BA 41 F2 8F AF 69 E7 3D
Plaintext:
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14
Checksum:
   45 EE 79 15 67 EE FC A3 7F 4A C1 E0 22 2D E8 0D
   43 C3 BF A0 66 99 67 2A

Sample pseudorandom function (PRF) invocations:
---------------------------------------

PRF input octet-string: "test" (0x74657374)

enctype aes128-cts-hmac-sha256-128:
base-key value / HMAC-SHA-256 key:
   37 05 D9 60 80 C1 77 28 A0 E8 00 EA B6 E0 D2 3C
HMAC-SHA-256 input message:
   00 00 00 01 70 72 66 74 65 73 74 00 00 00 01 00
PRF output:
   14 11 15 B0 A6 CB 9A 1D CB B4 C7 E2 5B 43 32 22
   52 DE 58 11 21 85 C5 DC F5 12 5E 7B 81 54 8D 39

enctype aes256-cts-hmac-sha384-192:
base-key value / HMAC-SHA-384 key:
   6D 40 4D 37 FA F7 9F 9D F0 D3 35 68 D3 20 66 98
   00 EB 48 36 47 2E A8 A0 26 D1 6B 71 82 46 0C 52
HMAC-SHA-384 input message:
   00 00 00 01 70 72 66 74 65 73 74 00 00 00 01 80
PRF output:
   31 0A 4B 5C D2 90 F7 04 33 B2 A1 A1 D0 93 FD F7
```

```
8C 6C 9D AE 5C AC D3 A7 BD 45 CB 67 44 41 99 43
0D 36 19 06 44 E8 A2 16 66 43 AE AD E9 63 87 52
```

Authors' Addresses

Michael J. Jenkins
National Security Agency

EMail: mjjenki@tycho.ncsc.mil

Michael A. Peck
The MITRE Corporation

EMail: mpeck@mitre.org

Kelley W. Burgin

Email: kelley.burgin@gmail.com