Network Working Group
Internet Draft
Intended Status: Informational
Expires: August 13, 2015

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

February 9, 2015

AES Encryption with HMAC-SHA2 for Kerberos 5
draft-ietf-kitten-aes-cts-hmac-sha2-06

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 August 13, 2015.

Copyright and License Notice

Copyright (c) 2015 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 16

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. All octets are expressed in big-endian order. The counter i is expressed as four octets and in this document is always 0x00000001 since there is only a single iteration of the PRF. The "Label" input to the NIST KDF is the constant supplied to this key derivation function. When deriving Kc, Ki, or Ke, the constant is the four octet key usage concatenated with 0x99, 0x55, or 0xAA respectively. When deriving the base-key, the

constant is the ASCII string "kerberos", also known as the byte string 0x6B65726265726F73. When deriving Kp, the constant is the ASCII string "prf", also known as the byte string 0x707266. The "Context" input is omitted. The length of the output key in bits (denoted as k) is also represented as four octets in big-endian order. Each application of the KDF only requires a single iteration of the PRF, so n = 1 in the notation of [SP800-108]. The purposes of the Kc, Ki, Ke, base-key, and Kp keys are described in Section 5.

In the following summary, | indicates concatenation. The random-to-key function is the identity function. The k-truncate function is defined in [RFC3961], Section 5.1.

When the encryption type is aes128-cts-hmac-sha256-128, the output key length k is 128 bits for all applications of KDF-HMAC-SHA2(key, constant) which is computed as follows:

```
K1 = HMAC-SHA-256(key, 00 00 00 01 | constant | 00 | 00 00 00 80)
KDF-HMAC-SHA2(key, constant) = random-to-key(k-truncate(K1))
```

When the encryption type is aes256-cts-hmac-sha384-192, the output key length k is 256 bits when deriving the base-key (from a passphrase as described in Section 4), Ke, and Kp. The output key length k is 192 bits when deriving Kc and Ki. KDF-HMAC-SHA2(key, constant) is computed as follows:

```
If deriving Kc or Ki (the constant ends with 0x99 or 0x55):
k = 192
K1 = HMAC-SHA-384(key, 00 00 00 01 | constant | 00 | 00 00 00 C0)
KDF-HMAC-SHA2(key, constant) = random-to-key(k-truncate(K1))

If deriving the base-key (the constant is "kerberos", the byte
string 0x6B65726265726F73), Ke (the constant ends with 0xAA),
or Kp (the constant is "prf", the byte string 0x707266):
k = 256
K1 = HMAC-SHA-384(key, 00 00 00 01 | constant | 00 | 00 00 01 00)
KDF-HMAC-SHA2(key, constant) = random-to-key(k-truncate(K1))
```

4. Key Generation from Pass Phrases

PBKDF2 [RFC2898] is used to derive the base-key from a passphrase and salt.

If no string-to-key parameters are specified, the default number of iterations is 32,768.

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

```
saltp = enctype-name | 0x00 | salt
tkey = random-to-key(PBKDF2(passphrase, saltp,
                        iter_count, keylength))
base-key = KDF-HMAC-SHA2(tkey, "kerberos") where "kerberos" is the
           byte 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 protocol-format 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: 00 00 80 00.

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.

```
Kc = KDF-HMAC-SHA2(base-key, usage | 0x99)
Ke = KDF-HMAC-SHA2(base-key, usage | 0xAA)
Ki = KDF-HMAC-SHA2(base-key, usage | 0x55)
```

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.

```
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]
cipherstate = the last full (128 bit) block of C
(i.e. the next-to-last block if the last block
is not a full 128 bits)
```

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 = the last full (128 bit) block of C
      (i.e. the next-to-last block if the last block
      is not a full 128 bits)

   pseudo-random function:
      If the enctype is aes128-cts-hmac-sha256-128:
      k = 128

      If the enctype is aes256-cts-hmac-sha384-192:
      k = 256

      Kp  = KDF-HMAC-SHA2(base-key, "prf")
      PRF = k-truncate(HMAC-SHA2(Kp, octet-string))

      where SHA2 is SHA-256 if the enctype is
      aes128-cts-hmac-sha256-128,
      and is SHA-384 if the enctype is aes256-cts-hmac-sha384-192.
```

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].

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 pass phrases or passwords. The salt and iteration count resist brute force and dictionary attacks, however, it is still important to choose or generate strong passphrases.

NIST guidance in section 5.3 of [SP800-38A] requires CBC initialization vectors be unpredictable. This specification does not formally comply with that guidance. However, the use of a confounder as the first block of plaintext 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 that a portion of the salt of at least 128 bits shall be randomly generated. Some known issues with including random values in Kerberos encryption type salt strings are:

* 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.

Further, 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.

* ktutil's add_entry command assumes the default salt.

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 use of SHA-384 and AES-256 with a 192-bit key provides only a 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
   Kp value (constant = 0x707266):
      9C 66 77 98 08 4F 16 82 1E 77 15 DD 5A A6 EB 71

   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
   Kp value (constant = 0x707266):
      5D 63 0D B7 EF DE 37 DE 9C 92 03 C5 2B D9 6C 77
      31 BE 1C 5B DD 50 DC 75 44 D9 60 AF F3 CC 23 04

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

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

   enctype aes128-cts-hmac-sha256-128:
   Kp value:
      9C 66 77 98 08 4F 16 82 1E 77 15 DD 5A A6 EB 71
   PRF output:
      3A CA 18 6C C1 26 56 76 5C FE B1 D2 2D 1C B1 36

   enctype aes256-cts-hmac-sha384-192:
   Kp value:
      5D 63 0D B7 EF DE 37 DE 9C 92 03 C5 2B D9 6C 77
      31 BE 1C 5B DD 50 DC 75 44 D9 60 AF F3 CC 23 04
   PRF output:
      01 72 03 F2 90 CD 16 6C D6 B2 BB 4F 18 7D 16 23
      6B 9A 4E D7 66 19 D8 11 6C 64 06 A3 37 E7 F9 08

   Sample encryptions (all using the default cipher state):
   -------------------------------------------------------

   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:
   9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
128-bit HMAC key:
   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:
   4E FD A6 52 4E 6B 56 B4 F2 12 61 FB FC 93 21 AB
128-bit HMAC key:
   29 1B 0C 37 73 D7 6E E6 BA 2C CF 1E 03 93 F6 3E
AES Output:
   AB 70 F4 BA 9D 76 55 AF 24 B5 76 E4 6E FB 7A 98
   F1 4B 93 65 9D 1B
Truncated HMAC Output:
   A0 C5 F4 7C AA 84 42 19 F9 08 AD ED EF 52 5B 71
Ciphertext:
   AB 70 F4 BA 9D 76 55 AF 24 B5 76 E4 6E FB 7A 98
   F1 4B 93 65 9D 1B A0 C5 F4 7C AA 84 42 19 F9 08
   AD ED EF 52 5B 71

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:
   FF 82 40 42 4B CC BA 05 56 50 C0 39 3B 83 DF 3B
128-bit HMAC key:
   ED 15 62 8B 45 35 8C BF 7F 50 E7 64 C2 6B 8A 1A
AES Output:
   E7 34 8E 74 86 E5 A7 87 0F 51 2E 65 CA C8 65 75
   78 26 FF C0 EA 5B 28 A8 B9 60 8B B3 08 CD E2 CC
Truncated HMAC Output:
   C1 85 4E F2 F3 4D 02 35 4E C7 AA 53 BE 03 BE D5
```

```
Ciphertext:
   E7 34 8E 74 86 E5 A7 87 0F 51 2E 65 CA C8 65 75
   78 26 FF C0 EA 5B 28 A8 B9 60 8B B3 08 CD E2 CC
   C1 85 4E F2 F3 4D 02 35 4E C7 AA 53 BE 03 BE D5

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:
   B5 9B 88 75 AD 5D CA FF F7 79 4D 93 F8 19 9D 79
128-bit HMAC key:
   0A 42 1D 72 2F 8F C2 D6 84 8B 1C DA D1 5A 49 C9
AES Output:
   C3 53 72 86 FF 9C FE 49 8D 2E FC FC 99 6D AC 2D
   52 CA 56 03 B3 E8 68 EA 1E 9C 54 E8 2A E5 CE 7A
   79 3E 21 09 7D
Truncated HMAC Output:
   5B 03 5D 78 A7 E9 84 75 EC 91 0C E3 7A A0 2A 7D
Ciphertext:
   C3 53 72 86 FF 9C FE 49 8D 2E FC FC 99 6D AC 2D
   52 CA 56 03 B3 E8 68 EA 1E 9C 54 E8 2A E5 CE 7A
   79 3E 21 09 7D 5B 03 5D 78 A7 E9 84 75 EC 91 0C
   E3 7A A0 2A 7D

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:
   0F A2 0D 7D 03 33 EE 65 16 2C DA 67 E7 AD 0D 3C
   5E 03 1F 3B 66 70 E0 31 28 2F AC C2 87 9C 21 C7
192-bit HMAC key:
   53 BF 30 6A 68 33 A3 25 18 FC B8 5F 63 1D 03 D5
   2E E3 1B 39 75 2F 57 ED
AES Output:
   FE 6A 55 14 F3 99 7C 8C AA F2 2D 8E EE 28 6D 7D
Truncated HMAC Output:
   81 1E AD AE DA 7F B9 75 AD 96 C0 07 5A 98 83 F9
   AC 3A AB 06 97 FC E8 5A
Ciphertext:
   FE 6A 55 14 F3 99 7C 8C AA F2 2D 8E EE 28 6D 7D
   81 1E AD AE DA 7F B9 75 AD 96 C0 07 5A 98 83 F9
   AC 3A AB 06 97 FC E8 5A
```

```
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:
   47 DA 4C A2 8B D1 C1 14 D5 50 7E 55 81 86 CA 4F
   DB A0 DA E5 B2 4F 6D 68 89 D5 3A FB F1 D0 B8 36
192-bit HMAC key:
   13 6B 5C 83 C9 53 AE 29 E2 C2 31 6A 7B 34 B8 C2
   AD 26 E4 66 7F AB 42 6E
AES Output:
   14 78 CF 26 BA 5E 7D 3A 9D C7 99 7A 80 10 76 2C
   74 3B D4 BC 22 EC
Truncated HMAC Output:
   17 2A B2 BB 12 B0 0D BE C2 BF E6 29 CF DD 62 EC
   3E 45 83 8F A9 FB AE 6E
Ciphertext:
   14 78 CF 26 BA 5E 7D 3A 9D C7 99 7A 80 10 76 2C
   74 3B D4 BC 22 EC 17 2A B2 BB 12 B0 0D BE C2 BF
   E6 29 CF DD 62 EC 3E 45 83 8F A9 FB AE 6E

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:
   5E A6 16 D8 FD A2 33 F1 B4 99 79 A4 B9 FA 01 D3
   21 B1 3D 6F BD 6E 3B B7 2E 54 B4 85 E2 36 AF 23
192-bit HMAC key:
   AD D3 8D C9 86 83 C5 CC 14 E3 C7 37 EA A7 06 47
   B3 19 71 0E 87 6A 38 77
AES Output:
   B6 0B 6A A6 00 C2 D8 4B 03 A6 1C 18 DD A7 05 F0
   FE 90 B9 36 B8 8C 4F EA 06 D7 1A 99 35 75 28 60
Truncated HMAC Output:
   2F E5 BD 6E 41 78 17 D6 2A D2 C9 CF 50 8D FA E1
   B3 C9 6F 4B 45 C1 9B 77
Ciphertext:
   B6 0B 6A A6 00 C2 D8 4B 03 A6 1C 18 DD A7 05 F0
   FE 90 B9 36 B8 8C 4F EA 06 D7 1A 99 35 75 28 60
   2F E5 BD 6E 41 78 17 D6 2A D2 C9 CF 50 8D FA E1
   B3 C9 6F 4B 45 C1 9B 77

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:
   B3 A8 02 E3 40 61 3E F1 E0 EC E9 1A 15 7C 59 12
   6F BD C4 B8 C2 4C 8D 0B 2E 5A 30 F0 1E 7E 34 88
192-bit HMAC key:
   FC 0B 49 9B 83 55 A3 2A C3 C9 AC B6 64 93 63 EB
   5D BB A4 25 1A 75 B2 0A
AES Output:
   4C F9 8B 5E DA 0D 94 9F B3 8E CD 67 DE 80 0F 79
   46 19 F9 EA CB 30 54 33 50 6B 9A D4 48 4B D9 5B
   E0 55 F5 69 EB
Truncated HMAC Output:
   7C F8 36 70 75 8C BF DA 31 3C FE F8 74 2B 11 74
   14 A7 DD 12 B4 96 64 2E
Ciphertext:
   4C F9 8B 5E DA 0D 94 9F B3 8E CD 67 DE 80 0F 79
   46 19 F9 EA CB 30 54 33 50 6B 9A D4 48 4B D9 5B
   E0 55 F5 69 EB 7C F8 36 70 75 8C BF DA 31 3C FE
   F8 74 2B 11 74 14 A7 DD 12 B4 96 64 2E


Sample checksums:
-----------------

Checksum type: hmac-sha256-128-aes128
128-bit HMAC key:
   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:
   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
```

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