AES Encryption with HMAC-SHA2 for Kerberos 5
draft-ietf-kitten-aes-cbc-hmac-sha2-00

Abstract

This document specifies two encryption types and two corresponding checksum types for Kerberos 5. The new types use AES in CBC mode with plaintext padding 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 January 20, 2014.

Copyright and License Notice

Copyright (c) 2013 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 Generation from Pass Phrases ............................. 3
4. Key Derivation Function ..................................... 4
5. Kerberos Algorithm Protocol Parameters ........................ 5
6. Checksum Parameters ........................................ 7
7. IANA Considerations .......................................... 7
8. Security Considerations ....................................... 7
   8.1. Random Values in Salt Strings ............................. 8
9. Acknowledgements ............................................. 8
10. References ................................................... 8
    10.1. Normative References .................................. 8
    10.2. Informative References ................................. 8
Appendix A. Test Vectors ........................................ 9
Authors’ Addresses ............................................. 15
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. The plaintext is padded to a multiple of the AES block size using the algorithm in Section 6.3 of [RFC5652]. 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 NSA’s Suite B Profile for Kerberos [suiteb-kerberos] which requires the use of SHA-256 or SHA-384 as the hash algorithm. Differences between the encryption and checksum types defined in this document and existing Kerberos encryption and checksum types are:

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

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

* The plaintext is padded so the resulting length is a multiple of the AES block length. This allows for AES encryption using CBC mode as defined in [SP800-38A] instead of using ciphertext stealing (CTS) mode.

* The random nonce used during content encryption is sent as part of the ciphertext, instead of using a confounder. This saves one encryption and decryption operation per message.

* The HMAC is calculated over the random nonce 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 Generation from Pass Phrases
The pseudorandom function used by PBKDF2 will be the SHA-256 or SHA-384 HMAC of the passphrase and salt. If the enctype is "aes128-cbc-hmac-sha256-128", then HMAC-SHA-256 is used as the PRF. If the enctype is "aes256-cbc-hmac-sha384-192", then HMAC-SHA-384 is used as the PRF.

The final key derivation step uses the algorithm KDF-HMAC-SHA2 defined below in Section 4.

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

To ensure that different long-term keys are used with different enctypes, we prepend the enctype name to the salt string, separated by a null byte. The enctype name is "aes128-cbc-hmac-sha256-128" or "aes256-cbc-hmac-sha384-192" (without the quotes). The user’s long-term key is derived as follows

```plaintext
saltp = enctype-name | 0x00 | salt
tkey = random-to-key(PBKDF2(passphrase, saltp, iter_count, keylength))
key = KDF-HMAC-SHA2(tkey, "kerberos") where "kerberos" is the byte string (0x6b65726265726f).
```

where the pseudorandom function used by PBKDF2 is HMAC-SHA-256 when the enctype is "aes128-cbc-hmac-sha256-128" and HMAC-SHA-384 when the enctype is "aes256-cbc-hmac-sha384-192", the value for keylength is the AES key length, and the algorithm KDF-HMAC-SHA2 is defined in Section 4.

4. Key Derivation Function

We use a key derivation function from Section 5.1 of [SP800-108] which uses the HMAC algorithm as the PRF. The counter i is expressed as four octets in big-endian order. The length of the output key in bits (denoted as k) is also represented as four octets in big-endian order. The "Label" input to the KDF is the usage constant supplied to the key derivation function, and the "Context" input is null. Each application of the KDF only requires a single iteration of the PRF, so n = 1 in the notation of [SP800-108].

In the following summary, | indicates concatenation. The random-to-key function is the identity function, as defined in Section 3. The k-truncate function is defined in [RFC3961], Section 5.1.

When the encryption type is aes128-cbc-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:

\[
K1 = \text{HMAC-SHA-256}(\text{key}, 00 \ 00 \ 00 \ 01 \ | \ \text{constant} \ | \ 0x00 \ | \ 00 \ 00 \ 00 \ 80) \\
\text{KDF-HMAC-SHA2} (\text{key, constant}) = \text{random-to-key}(k \text{-truncate}(K1))
\]

When the encryption type is aes256-cbc-hmac-sha384-192, the output key length \( k \) is 256 bits when computing the base-key and \( K_e \), and the output key length \( k \) is 192 bits when deriving \( K_c \) and \( K_l \). \text{KDF-HMAC-SHA2} (key, constant) is computed as follows:

If deriving \( K_c \) or \( K_l \) (the constant ends with 0x99 or 0x55):

\[
k = 192 \\
K1 = \text{HMAC-SHA-384}(\text{key}, 00 \ 00 \ 00 \ 01 \ | \ \text{constant} \ | \ 0x00 \ | \ 00 \ 00 \ 00 \ C0) \\
\text{KDF-HMAC-SHA2} (\text{key, constant}) = \text{random-to-key}(k \text{-truncate}(K1))
\]

Otherwise (if deriving \( K_e \) or deriving the base-key from a passphrase as described in Section 3):

\[
k = 256 \\
K1 = \text{HMAC-SHA-384}(\text{key}, 00 \ 00 \ 00 \ 01 \ | \ \text{constant} \ | \ 0x00 \ | \ 00 \ 00 \ 01 \ 00) \\
\text{KDF-HMAC-SHA2} (\text{key, constant}) = \text{random-to-key}(k \text{-truncate}(K1))
\]

The constants used for key derivation are the same as those used in the simplified profile.

### 5. Kerberos Algorithm Protocol Parameters

Each encryption will use a 16-octet nonce generated at random by the message originator. The initialization vector (IV) used by AES is obtained by xoring the random nonce with the cipherState.

CBC mode [SP800-38A] requires the plaintext length be a multiple of the AES block size, so the plaintext is padded using the algorithm in Section 6.3 of [RFC5652].

The ciphertext is the concatenation of the random nonce, the output of AES in CBC mode, and the HMAC of the nonce concatenated with the AES output. The HMAC is computed using either SHA-256 or SHA-384. 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 remainder, and then decrypting the remainder if the HMAC is correct.

The following parameters apply to the encryption types aes128-cbc-hmac-sha256-128 and aes256-cbc-hmac-sha384-192.
protocol key format: as defined in Section 2.
specific key structure: three protocol-format keys: \{ Kc, Ke, Ki \}.
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 3.
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 4. 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 random nonce.
initial cipherState: all bits zero.

encryption function: as follows, where E() is AES encryption in CBC mode, h is the size of truncated HMAC, and c is the AES block size.

\[
N = \text{random nonce of length c (128 bits)}
\]
\[
IV = N \oplus \text{cipherState}
\]
\[
pad = \text{Shortest string of non-zero length to bring the plaintext to a length that is a multiple of c. The value of each added octet equals the number of octets that are added.}
\]
\[
C = E(Ke, \text{plaintext} \mid pad, IV)
\]
\[
H = \text{HMAC(Ki, N} \mid C)
\]
\[
ciphertext = N \mid C \mid H[1..h]
\]
\[
cipherState = N
\]
decryption function: as follows, where D() is AES encryption in CBC mode, and h is the size of truncated HMAC.

\[
(N, C, H) = \text{ciphertext}
\]
\[
\text{if } H \neq \text{HMAC(Ki, N} \mid C)[1..h]
\]
\[
\text{stop, report error}
\]
\[
IV = N \oplus \text{cipherState}
\]
\[
P \mid pad = D(Ke, C, IV)
\]
\[
cipherState = N
\]
pseudo-random function:
\[ K_p = \text{KDF-HMAC-SHA2}(\text{protocol-key}, \text{"prf"}) \]
\[ \text{PRF} = \text{HMAC}(K_p, \text{octet-string}) \]

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-cbc-hmac-sha256-128 and aes256-cbc-hmac-sha384-192, respectively.

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

\[ \text{get_mic: HMAC}(K_c, \text{message})[1..h]. \]
\[ \text{verify_mic: get_mic and compare.} \]

7. IANA Considerations

IANA is requested to assign:

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

<table>
<thead>
<tr>
<th>Etype</th>
<th>encryption type</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD1</td>
<td>aes128-cbc-hmac-sha256-128</td>
<td>[this document]</td>
</tr>
<tr>
<td>TBD2</td>
<td>aes256-cbc-hmac-sha384-192</td>
<td>[this document]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Sumtype</th>
<th>Checksum type</th>
<th>Size</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD3</td>
<td>hmac-sha256-128-aes128</td>
<td>16</td>
<td>[this document]</td>
</tr>
<tr>
<td>TBD4</td>
<td>hmac-sha384-192-aes256</td>
<td>24</td>
<td>[this document]</td>
</tr>
</tbody>
</table>

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.

8.1. Random Values in Salt Strings

NIST guidance in Section 5.1 of [SP800-132] requires the salt used as input to the PBKDF to contain at least 128 bits of random. Some known issues with including random values in Kerberos encryption type salt strings are:

* Cross-realm TGTs are currently 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.

* 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.

* Current implementations of password history checking will not work.

* ktutil’s add_entry command assumes the default salt.

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


10.2. Informative References

Appendix A.  Test Vectors

Sample results for string-to-key conversion:

--------------------------------------------

Iteration count = 32768
Pass phrase = "password"

Saltp for creating 128-bit master key:
   61 65 73 31 32 38 2D 63 62 63 2D 68 6D 61 63 2D
   73 68 61 32 35 36 2D 31 32 38 00 10 DF 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-cbc-hmac-sha256-128" | 0x00 |
random 16 byte valid UTF-8 sequence | "ATHENA.MIT.EDUraburn")

128-bit master key:
   C3 19 22 E2 EA 3A 67 05 E0 B9 AC 57 08 82 48 28

Saltp for creating 256-bit master key:
   61 65 73 32 35 36 2D 63 62 63 2D 68 6D 61 63 2D
   73 68 61 33 38 34 2D 31 39 32 00 10 DF 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-cbc-hmac-sha384-192" | 0x00 |
random 16 byte valid UTF-8 sequence | "ATHENA.MIT.EDUraburn")

256-bit master key:
   77 73 83 E7 C4 76 1D CE FC 5B D8 F8 A7 28 37 8A
   5E 63 BC B2 0E B9 A2 BB C5 1E 73 56 8A FC CD E6
Sample results for key derivation:
----------------------------------

enctype aes128-cbc-hmac-sha256-128:

128-bit master 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-cbc-hmac-sha384-192:

256-bit master 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
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 2D 10 D5 C7 30 12 B6
  22 C4 D0 0F FC 23 ED 1F

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

Plaintext: (empty)
Nonce:
  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:
  9E 30 E1 7A 01 BC E8 5B 59 90 C8 90 1A 55 1D 8C
HMAC Output:
  0C 80 06 07 A4 6E 35 2C A7 73 CE 52 69 51 63 57
Ciphertext:
  7E 58 95 EA F2 67 24 35 BA D8 17 F5 45 A3 71 48
  9E 30 E1 7A 01 BC E8 5B 59 90 C8 90 1A 55 1D 8C
  0C 80 06 07 A4 6E 35 2C A7 73 CE 52 69 51 63 57
Plaintext: (length less than block size)
  00 01 02 03 04 05
Nonce:
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:
2B E8 63 D7 B1 D4 F0 4D 95 F2 17 D6 9E C2 14 23
HMAC Output:
5F D1 CB B9 C0 6E 42 6E F9 95 05 B5 FB 42 6F 6A
Ciphertext:
7B CA 28 5E 2F D4 13 0F B5 5B 1A 5C 83 BC 5B 24
2B E8 63 D7 B1 D4 F0 4D 95 F2 17 D6 9E C2 14 23
5F D1 CB B9 C0 6E 42 6E F9 95 05 B5 FB 42 6F 6A
Plaintext: (length equals block size)
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Nonce:
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:
AD 5D 0C E8 93 48 A8 16 07 11 09 75 6A 83 FB 09
D2 3F 29 30 68 F9 D4 E5 1F B8 92 B0 61 C7 43 BF
HMAC Output:
3A 40 51 A4 8B 7A 11 B3 91 F1 36 67 98 16 24 AD
Ciphertext:
56 AB 21 71 3F F6 2C 0A 14 57 20 0F 6F A9 94 8F
AD 5D 0C E8 93 48 A8 16 07 11 09 75 6A 83 FB 09
D2 3F 29 30 68 F9 D4 E5 1F B8 92 B0 61 C7 43 BF
3A 40 51 A4 8B 7A 11 B3 91 F1 36 67 98 16 24 AD
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
Nonce:
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:
DA A3 99 2E 39 5C 5D E1 34 EB 1A CC 73 8D CE 02
35 B9 D6 5A 63 0B 8D 84 BC 78 E9 38 75 79 5E DF
HMAC Output:
CF 68 74 07 12 22 6C 61 C1 E4 A6 78 A9 7C 86 60
Ciphertext:
A7 A4 E2 9A 47 28 CE 10 66 4F B6 4E 49 AD 3F AC
DA A3 99 2E 39 5C 5D E1 34 EB 1A CC 73 8D CE 02
35 B9 D6 5A 63 OB 8D 84 BC 78 E9 38 75 79 5E DF
CF 68 74 07 12 22 6C 61 C1 E4 A6 78 A9 7C 86 60

Plaintext: (empty)

Nonce:
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 5F 57 ED

AES Output:
73 1E 56 A3 D9 DA 70 87 5C 74 C7 67 73 C2 F7 EB

HMAC Output:
FA F7 49 55 33 7E 20 98 C4 B4 F7 8F 35 5B 8A B9
72 6D 40 AC F3 5D B3 7B

Ciphertext:
F7 64 E9 FA 15 C2 76 47 8B 2C 7D 0C 4E 5F 58 E4
73 1E 56 A3 D9 DA 70 87 5C 74 C7 67 73 C2 F7 EB
FA F7 49 55 33 7E 20 98 C4 B4 F7 8F 35 5B 8A B9
72 6D 40 AC F3 5D B3 7B

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

Nonce:
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:
EF DE 87 A1 14 2D B5 C7 4A 42 52 A7 A7 77 5A 3E

HMAC Output:
45 02 19 E4 A8 C6 3E 8F E6 DB F5 08 78 E4 28 40
E9 36 DD 0A 66 1C A9 9C

Ciphertext:
B8 0D 32 51 C1 F6 47 14 94 25 6F FE 71 2D 0B 9A
EF DE 87 A1 14 2D B5 C7 4A 42 52 A7 A7 77 5A 3E
45 02 19 E4 A8 C6 3E 8F E6 DB F5 08 78 E4 28 40
E9 36 DD 0A 66 1C A9 9C

Plaintext: (length equals block size)
Nonce:
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:
E4 09 FF 7A 93 60 E9 72 7B 3F 88 35 28 73 E0 CF
B3 21 90 09 69 7D 79 6A 51 9C A3 86 DF 84 5D AD

HMAC Output:
60 75 75 AA D0 05 9F 9A C8 16 EA E0 B9 B5 00 2E
42 33 AA 53 89 9F AB 39

Ciphertext:
53 BF 8A 0D 10 52 65 D4 E2 76 42 86 24 CE 5E 63
E4 09 FF 7A 93 60 E9 72 7B 3F 88 35 28 73 E0 CF
B3 21 90 09 69 7D 79 6A 51 9C A3 86 DF 84 5D AD
60 75 75 AA D0 05 9F 9A C8 16 EA E0 B9 B5 00 2E
42 33 AA 53 89 9F AB 39

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

Nonce:
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:
F6 2D D7 FF 39 A8 EE D2 4C 5A 8F CF 84 15 71 1C
F5 05 05 2F 9B AD 75 C8 27 9D 05 D4 81 CF A9 73

HMAC Output:
DB 3B C2 37 0F 9D A6 F1 F7 99 32 A0 A6 4F 7A 7A
BD B9 B3 35 47 DD 9B 62

Ciphertext:
76 3E 65 36 7E 86 4F 02 F5 51 53 C7 E3 B5 8A F1
F6 2D D7 FF 39 A8 EE D2 4C 5A 8F CF 84 15 71 1C
F5 05 05 2F 9B AD 75 C8 27 9D 05 D4 81 CF A9 73
DB 3B C2 37 0F 9D A6 F1 F7 99 32 A0 A6 4F 7A 7A
BD B9 B3 35 47 DD 9B 62

Sample checksums:
-----------------
Checksum type: hmac-sha256-128-aes128
128-bit master key:
37 05 D9 60 80 C1 77 28 A0 E8 00 EA B6 E0 D2 3C
128-bit HMAC key (Kc, key usage 2):
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
256-bit master 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
192-bit HMAC key (Kc, key usage 2):
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