Network Working Group
Internet Draft
Intended Status: Informational
Expires: October 21, 2013

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

April 19, 2013

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

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 October 21, 2013.

Copyright and License Notice

Table of Contents

1. Introduction 3
2. Conventions used in this Document 3
3. Protocol Key Representation 3
4. Key Generation from Pass Phrases 3
5. Key Derivation Function 4
6. Kerberos Algorithm Protocol Parameters 5
7. IANA Considerations 8
8. Security Considerations 8
9. References 9
9.1. Normative References 9
9.2. Informative References 9
Appendix A. Test Vectors 10
Authors' Addresses 12

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 new types conform to the framework specified in [RFC3961], but do not use the simplified profile.

The new encryption types use AES in CTS mode (CBC mode with ciphertext stealing) similar to [RFC3962] but with several variations.

The new types use the PBKDF2 algorithm for key generation from strings, with a modification to the use in [RFC3962] that the pseudorandom function used by PBKDF2 is HMAC-SHA-256 or HMAC-SHA-384 instead of HMAC-SHA-1.

The new types use key derivation to produce keys for encryption, integrity protection, and checksum operations as in [RFC3962]. However, a key derivation function from [SP800-108] which uses the SHA-256 or SHA-384 hash algorithm is used in place of the DK key derivation function used in [RFC3961].

The new types use the HMAC algorithm with a hash from the SHA-2 family for integrity protection and checksum operations.

2. Conventions used in this Document

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].

3. 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).

4. Key Generation from Pass Phrases

We use a variation on the key generation algorithm specified in Section 4 of [RFC3962] with the following changes:

* The pseudorandom function used by PBKDF2 will be the SHA-256 or SHA-384 HMAC of the passphrase and salt, instead of the SHA-1 HMAC of the passphrase and salt. If the enctype is "aes128-cts-hmac-sha256-128", then HMAC-SHA-256 is used as the PRF. If the enctype is "aes256-cts-hmac-sha384-192", then HMAC-SHA-384 is used as the PRF.

* The salt MUST contain at least 128 random bits as required in Section 5.1 of [SP800-132]. It MAY also contain other information such as the principal's realm and name components.

* The final key derivation step uses the algorithm KDF-HMAC-SHA2 defined below in Section 5 instead of the DK function.

* If no string-to-key parameters are specified, the default number of iterations is raised to 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-cts-hmac-sha256-128" or "aes256-cts-hmac-sha384-192" (without the quotes). The user's long-term key is derived as follows

```
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 {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, and the algorithm KDF-HMAC-SHA2 is defined in Section 5.

5. 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. In the following summary, | indicates concatenation. The random-to-key function is the identity function, as defined in Section 6. 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:

```
n = 1
K1 = HMAC-SHA-256(key, 00 00 00 01 | constant | 0x00 | 00 00 00 80)
DR(key, constant) = k-truncate(K1)
KDF-HMAC-SHA2(key, constant) = random-to-key(DR(key, constant))
```

When the encryption type is aes256-cts-hmac-sha384-192, the output key length k is 256 bits when computing the base-key and Ke, and 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
n = 1
K1 = HMAC-SHA-384(key, 00 00 00 01 | constant | 0x00 | 00 00 00 C0)
DR(key, constant) = k-truncate(K1)
KDF-HMAC-SHA2(key, constant) = random-to-key(DR(key, constant))

Otherwise (if deriving Ke or deriving the base-key from a
```

passphrase as described in Section 4):

```
k = 256
n = 1
K1 = HMAC-SHA-384(key, 00 00 00 01 | constant | 0x00 | 00 00 01 00)
DR(key, constant) = k-truncate(K1)
KDF-HMAC-SHA2(key, constant) = random-to-key(DR(key, constant))
```

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

6. Kerberos Algorithm Protocol Parameters

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

The key-derivation function described in the previous section is used to produce the three intermediate keys. Typically, CBC mode [SP800-38A] requires the input be padded to a multiple of the encryption algorithm block size, which is 128 bits for AES. However, to avoid ciphertext expansion, we use the CBC-CS3 variant to CBC mode defined in [SP800-38A+] (this mode is also referred to as CTS). Note that [SP800-38A+] requires the plaintext length to be greater than or equal to the block size.

Each encryption will use a freshly generated 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.

The ciphertext is the concatenation of the random nonce, the output of AES in CBC-CS3 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 SHA-256 is truncated to 128 bits and the output of 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 encryption and checksum mechanisms below use the following

notation from [RFC3961].

```
HMAC output size, h
message block size, m
encryption/decryption functions, E and D
cipher block size, c
```

Encryption Mechanism for AES-CTS-HMAC-SHA2

protocol key format	128- or 256-bit string
specific key structure	Three protocol-format keys: { Kc, Ke, Ki }.
required checksum mechanism	As defined below.
key-generation seed length	key size (128 or 256 bits)
cipher state	Random nonce of length c (128 bits)
initial cipher state	All bits zero
encryption function	N = random nonce of length c (128 bits) IV = N + cipherState (+ denotes XOR) C = E(Ke, plaintext, IV) using CBC-CS3-Encrypt defined in [SP800-38A+] H = HMAC(Ki, N | C) ciphertext = N | C | H[1..h] cipherState = N

decryption function	(N, C, H) = ciphertext if (H != HMAC(Ki, N | C)[1..h]) stop, report error IV = N + cipherState (+ denotes XOR) P = D(Ke, C, IV) using CBC-CS3-Decrypt defined in [SP800-38A+] cipherState = N

pseudo-random function	Kp = KDF-HMAC-SHA2(protocol-key, "prf") PRF = HMAC(Kp, octet-string)
key generation functions:	
string-to-key function	tkey = random-to-key(PBKDF2(passphrase, saltp, iter_count, keylength)) base-key = KDF-HMAC-SHA2(tkey, "kerberos") where the pseudorandom function used by PBKDF2 is HMAC-SHA-256 or HMAC-SHA-384 as described 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 5. 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);

Checksum Mechanism for AES-CTS-HMAC-SHA2

associated cryptosystem	AES-128-CTS or AES-256-CTS as appropriate
get_mic	HMAC(Kc, message)[1..h]
verify_mic	get_mic and compare

Using this profile with each key size gives us two each of encryption and checksum algorithm definitions.

+----------+

encryption types		
type name	etype value	key size
aes128-cts-hmac-sha256-128	TBD1	128
aes256-cts-hmac-sha384-192	TBD2	256

checksum types		
type name	sumtype value	length
hmac-sha256-128-aes128	TBD3	128
hmac-sha384-192-aes256	TBD4	192

These checksum types will be used with the corresponding encryption types defined above.

7. IANA Considerations

IANA is requested to assign:

1. 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]

2. 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. NIST Special Publication 800-90 [SP800-90] and [RFC4086] offer 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.

9. References

9.1. Normative References

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

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

[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.

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

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

9.2. Informative References

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

[SP800-90] National Institute of Standards and Technology, Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised), NIST

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

[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 master 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 F3 60 61 DC E2
   E1 B3 59 00 83 87 46 B8 78 2F 1D 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 |
 16 random bytes | "ATHENA.MIT.EDUraeburn")
128-bit master key:
   37 05 D9 60 80 C1 77 28 A0 E8 00 EA B6 E0 D2 3C

Saltp for creating 256-bit master 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 F3 60 61 DC E2
   E1 B3 59 00 83 87 46 B8 78 2F 1D 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 |
 16 random bytes | "ATHENA.MIT.EDUraeburn")
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
```

Sample results for key derivation:

```
enctype aes128-cts-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-cts-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
   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 (using the default cipher state):

128-bit master key:
   37 05 D9 60 80 C1 77 28 A0 E8 00 EA B6 E0 D2 3C
128-bit AES key (Ke, key usage 2):
   9B 19 7D D1 E8 C5 60 9D 6E 67 C3 E3 7C 62 C7 2E
128-bit HMAC key (Ki, key usage 2):
   9F DA 0E 56 AB 2D 85 E1 56 9A 68 86 96 C2 6A 6C
Plaintext:
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14
IV | Ciphertext | Authentication Tag:
   8D 32 50 F6 36 AB 81 02 BE 6F AB 1E 57 D8 F8 17
   13 64 FB 39 DC C0 E3 D9 83 A7 DB 5B 4B 9F FB CA
   42 F6 65 88 29 F2 1F C8 95 75 AE 93 C7 57 18 AB
   3C 7C FB 28 E1

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
256-bit AES key (Ke, key usage 2):
   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, key usage 2):
   69 B1 65 14 E3 CD 8E 56 B8 20 10 D5 C7 30 12 B6
   22 C4 D0 0F FC 23 ED 1F
Plaintext:
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14
IV | Ciphertext | Authentication Tag:
   8D 32 50 F6 36 AB 81 02 BE 6F AB 1E 57 D8 F8 17
   50 CB FF DC DF 38 69 D7 0B EA FF C3 2C 47 0B C6
   5B 72 C3 37 2D 6E D7 B3 47 E9 0B BD 8F 31 F5 79
   58 F9 69 50 BA A1 41 64 6E 65 6C F6 7C
```


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

Kelley W. Burgin
National Security Agency

EMail: kwburgi@tycho.ncsc.mil

Michael A. Peck
The MITRE Corporation

EMail: mpeck@mitre.org