Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: February 19, 2011

S. Kanno
NTT Software Corporation
L. Howard
PADL Software Ltd
T. Yu
T. Hardjono
MIT Kerberos Consortium

August 18, 2010

Kerberos Support for Camellia Cipher in CCM Mode
draft-kanno-krbwg-camellia-ccm-03

Abstract

This draft proposes the Kerberos (v5) support for the Camellia Cipher in Counter with CBC-MAC (CCM) mode.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines encryption key and checksum types for Kerberos (v5) using the Camellia algorithm in Counter with CBC-MAC (CCM) Mode [SP800-38C]. The Camellia cipher was developed by NTT and Mitsubishi Electric Corporation in 2000. These new types support 128-bit block encryption and key sizes of 128 or 256 bits. The Camellia algorithm and its properties are described in [RFC3713].

There are a number of motivations to providing support for Camellia in Kerberos v5. Among others, it is desirable to provide an alternate cipher should weaknesses be discovered in the AES and SHA-256 algorithms which are predominant today. Additionally, due to the international user-base of Kerberos, supporting additional ciphers in key markets allows easier adoption and deployment of Kerberos in those regions.

Because the encryption types use the CCM mode, they do not rely on a hash algorithm to ensure message integrity. To preserve this property, the corresponding checksum types use the CMAC algorithm [SP800-38B].

For key derivation, the encryption and checksum types use KDF in feedback mode as described in [SP800-108], with CMAC as the underlying pseudo-random function.

2. Requirements Language

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 profile in [RFC3961] treats keys and random octet strings as conceptually different. But since the Camellia key space is dense, we can use any bit string of appropriate length as a key. We use the byte representation for the key described in [RFC3713], where the first bit of the bit string is the high bit of the first byte of the byte string (octet string) representation.

4. Key Generation from Pass Phrases or Random Data

Given the above format for keys, we can generate keys from the appropriate amounts of random data (128 or 256 bits) by simply

copying the input string. To generate an encryption key from a pass phrase and salt string, we use a slight variation on the equivalent algorithm described in section 4 of [RFC3962]. To ensure that different long-term keys are used with Camellia and AES, we prepend the enctype name to the salt string, separated by a null byte. The enctype name is "camellia128-ccm-128" or "camellia256-ccm-128" (without the quotes).

```
saltp = enctype-name | 0x00 | salt

tkey = random2key(PBKDF2(passphrase, saltp, iter_count, keylength))

key = DK(tkey, "kerberos")
```

The pseudo-random function used by PBKDF2 is unchanged from section 4 of [RFC3962], as is the default iteration count if no string-to-key parameters are supplied.

5. Kerberos Algorithm Profile Parameters

This is a summary of the parameters to be used with the simplified algorithm profile described in [RFC3961].

```
                          Cryptosystem from CCM Profile
------------------------------------------------------------------------
protocol key format       As given.

specific key structure    Two protocol-format keys: { Kc, Ke }.

key-generation seed       As given.
length

required checksum         As defined below.
mechanism

cipher state              counter index i, expressed as q octets in
                          big-endian order

initial cipher state      i = 0

encryption function       adata = associated data
                          adata_pad = shortest string of zero octets to
                                      bring adata to a length that is a
                                      multiple of the block size
                          plaintext_pad = shortest string of zero octets
                                          to bring plaintext to a length
```

```
                                        that is a multiple of the
                                        block size
                          N = random nonce of length n
                          Q = binary representation of octet length of
                              plaintext of length q
                          i = counter index
                          m = number of blocks
                          B0 = Flags | N | Q
                          Ctr0 = Flags | N | oldstate.i
                          T = CBC-MAC(Ke, B0 | adata_len |
                                     adata_pad | pad | plaintext | pad)
                          (H1, Ctr1) = E(Ke, T, Ctr0)
                          (C1, Ctrm) = E(Ke, plaintext, Ctr1)
                          ciphertext = N | C1 | H1
                          newstate.i = Ctrm.i

decryption function       (N,C1,H1) = ciphertext
                          Ctr0 = Flags | N | oldstate.i
                          (T, Ctr1) = D(Ke, H1, Ctr0)
                          (P1, Ctrm) = D(Ke, C1, Ctr1)
                          if (T != CBC-MAC(Ke, B0 | adata_len |
                                   adata | adata_pad | plaintext | pad))
                             report error
                          newstate.i = Ctrm.i

default string-to-key     As given.
params

pseudo-random function    PRF = CMAC(DK(protocol-key, prfconstant),
                                     octet-string)

                          The "prfconstant" used in the PRF operation
                          is the three-octet string "prf".
                          CMAC is defined in [SP800-38B].  The
                          underlying cipher is the associated
                          cryptosystem of the encryption type.

key generation functions:

string-to-key function    As given.

random-to-key function    As given.

key-derivation function   The "well-known constant" used for the DK
                          function is the key usage number, expressed as
                          four octets in big-endian order, followed by
                          one octet indicated below.
```

```
                          Kc = DK(base-key, usage | 0x99);
                          Ke = DK(base-key, usage | 0xCC);

                          where

                          DK(Key, Constant) = random-to-key(DR(Key,
                                                         Constant))
                          K0 = zeros
                          Ki = CMAC(Key, K(i-1) | i | Constant |
                                                 0x00 | Length(DK))
                          DR(Key, Constant) = k-truncate(K1 | K2 |
                                                        K3 | K4 ...)

                          i, Length expressed as four octets in big-
                          endian order
```

```
                    Checksum Mechanism from CCM Profile
             --------------------------------------------------
             associated cryptosystem   As defined above.

             get_mic                   CMAC(Kc, message)

             verify_mic                get_mic and compare
```

Figure 1

```
   +--------------------------------------------------------------------+
   |               protocol key format        128- or 256-bit string    |
   |                                                                    |
   |            string-to-key function        PBKDF2+DK with variable   |
   |                                          iteration count (see      |
   |                                          above) and salt given by  |
   |                                          type-name | 0x00 | salt   |
   |                                                                    |
   |                                          type-name is "camellia    |
   |                                          128-ccm-128" or "camellia |
   |                                          256-ccm-128" (without the |
   |                                          quotes). salt is the      |
   |                                          original input to the     |
   |                                          string-to-key function.   |
   |                                                                    |
   |  default string-to-key parameters        00 00 10 00               |
   |                                                                    |
   |        key-generation seed length        key size                  |
   |                                                                    |
   |            random-to-key function        identity function         |
   |                                                                    |
```

nonce length, n	12 octets (96 bits)
tag length, t	16 octets (128 bits)
counter length, q	3 octets (24 bits)
message block size, m	1 octet
encryption/decryption functions, E and D	Camellia in CTR mode (cipher block size 16 octets), with counter block as cipher state

encryption types		
type-name	etype value	key size
camellia128-ccm-128	TBD	128
camellia256-ccm-128	TBD	256

checksum types		
type-name	sumtype value	length
cmac-128-camellia128	TBD	128
cmac-128-camellia256	TBD	128

Figure 2

6. Assigned numbers

TBD

7. IANA Considerations

Kerberos encryption and checksum type values used in section 7 were previously reserved in [RFC3961] for the mechanisms defined in this document. The registries have been updated to list this document as the reference.

8. Security Considerations

At the time of writing this document, there are no known weak keys for Camellia, and no security problem has been found on Camellia (see [NESSIE], [CRYPTREC], and [LNCS]).

The CCM mode requires a unique nonce for each message. If two messages use the same nonce, the XOR of the plain texts of the messages can be recovered without the key, compromising the confidentiality of the messages. Kerberos can only probabilistically ensure nonce uniqueness by choosing random nonce values. Since the length of the nonce is 96 bits, the probability of a collision becomes significant as the number of observed messages approaches 2^48.

CCM was chosen over GCM partly in order to minimize the impact of a nonce collision. Under GCM, a nonce collision results not only in a loss of confidentiality of the plaintexts, but also in the ability to construct forged messages.

9. Test Vectors

TBD

10. Acknowledgements

We would like to thank Greg Hudson for the review and corrections to this draft.

11. References

11.1. Normative References

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

[RFC3713] Matsui, M., Nakajima, J., and S. Moriai, "A Description of the Camellia Encryption Algorithm", RFC 3713, April 2004.

[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.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[SP800-108]
Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, October 2009, <http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf>.

[SP800-38B]
Dworkin, M., "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", NIST Special Publication 800-38B, May 2005, <http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf>.

[SP800-38C]
Dworkin, M., "Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality", NIST Special Publication 800-38C, July 2007, <http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf>.

11.2. Informative References

[CRYPTREC]
Information-technology Promotion Agency (IPA), "Cryptography Research and Evaluation Committees", <http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html>.

[ISO-18033-3]
International Standards Organization, "Information technology - Security techniques - Encryption algorithms -- Part 3: Block ciphers (AES, Camellia, SEED)", July 2010.

[LNCS]   Mala, H., Shakiba, M., and M. Dakhil-alian, "New Results on Impossible Differential Cryptanalysis of Reduced Round Camellia-128", LNCS 5867, November 2009, <http://www.springerlink.com/content/e55783u422436g77/>.

[MIT-Athena]
Steiner, J., Neuman, B., and J. Schiller, "Kerberos: An Authentication Service for Open Network Systems. In Proceedings of the Winter 1988 Usenix Conference. February.", 1988.

[NESSIE]   "The NESSIE project (New European Schemes for Signatures,

              Integrity and Encryption)",
              <http://www.cosic.esat.kuleuven.be/nessie/>.

   [RFC1510] Kohl, J. and B. Neuman, "The Kerberos Network
              Authentication Service (V5)", RFC 1510, September 1993.

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

   [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC
              Text on Security Considerations", BCP 72, RFC 3552,
              July 2003.

Appendix A. Additional Stuff

This becomes an Appendix.

Authors' Addresses

Satoru Kanno
NTT Software Corporation

Phone: +81-45-212-9803
Email: kanno.satoru@po.ntts.co.jp

Luke Howard
PADL Software Ltd

Email: lukeh@padl.com

Tom Yu
MIT Kerberos Consortium

Email: tlyu@mit.edu

Thomas Hardjono
MIT Kerberos Consortium

Email: hardjono@mit.edu