S/MIME Working Group   S. Moriai
Internet Draft   Sony Computer Entertainment Inc.
Document: draft-ietf-smime-camellia-05.txt   A. Kato
Expires: February 2004   NTT Software Corporation
                                                             August 2003

Use of the Camellia Encryption Algorithm in CMS

Abstract
This document specifies the conventions for using the Camellia encryption algorithm for encryption with the Cryptographic Message Syntax (CMS).

1. Introduction

This document specifies the conventions for using the Camellia encryption algorithm [CamelliaSpec][CamelliaID] for encryption with the Cryptographic Message Syntax (CMS) [CMS]. The relevant object identifiers (OIDs) and processing steps are provided so that Camellia may be used in the CMS specification (RFC 3369, RFC 3370) for content and key encryption.

Note:
This work was done when the first author worked for NTT.

1.1 Camellia

Camellia was jointly developed by Nippon Telegraph and Telephone

Corporation and Mitsubishi Electric Corporation in 2000. Camellia specifies the 128-bit block size and 128-, 192-, and 256-bit key sizes, the same interface as the Advanced Encryption Standard (AES). Camellia is characterized by its suitability for both software and hardware implementations as well as its high level of security. From a practical viewpoint, it is designed to enable flexibility in software and hardware implementations on 32-bit processors widely used over the Internet and many applications, 8-bit processors used in smart cards, cryptographic hardware, embedded systems, and so on [CamelliaTech]. Moreover, its key setup time is excellent, and its key agility is superior to that of AES.

Camellia has been scrutinized by the wide cryptographic community during several projects for evaluating crypto algorithms. In particular, Camellia was selected as a recommended cryptographic primitive by the EU NESSIE (New European Schemes for Signatures, Integrity and Encryption) project [NESSIE] and also included in the list of cryptographic techniques for Japanese e-Government systems which were selected by the Japan CRYPTREC (Cryptography Research and Evaluation Committees) [CRYPTREC].

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [RFC2119].

2. Object Identifiers for Content and Key Encryption

This section provides the OIDs and processing information necessary for Camellia to be used for content and key encryption in CMS.

Camellia is added to the set of optional symmetric encryption algorithms in CMS by providing two classes of unique object identifiers (OIDs). One OID class defines the content encryption algorithms and the other defines the key encryption algorithms. Thus a CMS agent can apply Camellia either for content or key encryption by selecting the corresponding object identifier, supplying the required parameter, and starting the program code.

2.1 OIDs for Content Encryption

Camellia is added to the set of symmetric content encryption algorithms defined in [CMSALG]. The Camellia content-encryption algorithm, in Cipher Block Chaining (CBC) mode, for the three different key sizes are identified by the following object identifiers:

```
   id-camellia128-cbc OBJECT IDENTIFIER ::=
           { iso(1) member-body(2) 392 200011 61 security(1)
             algorithm(1) symmetric-encryption-algorithm(1)
             camellia128-cbc(2) }
```

```
   id-camellia192-cbc OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) symmetric-encryption-algorithm(1)
         camellia192-cbc(3) }

   id-camellia256-cbc OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) symmetric-encryption-algorithm(1)
         camellia256-cbc(4) }
```

The AlgorithmIdentifier parameters field MUST be present, and the parameters field MUST contain the value of IV:

```
   CamelliaCBCParameter ::= CamelliaIV  --  Initialization Vector

   CamelliaIV ::= OCTET STRING (SIZE(16))
```

The plain text is padded according to Section 6.3 of [CMS].

2.2 OIDs for Key Encryption

The key-wrap/unwrap procedures used to encrypt/decrypt a Camellia content-encryption key (CEK) with a Camellia key-encryption key (KEK) are specified in Section 3. Generation and distribution of key-encryption keys are beyond the scope of this document.

The Camellia key-encryption algorithm has the following object identifier:

```
  id-camellia128-wrap OBJECT IDENTIFIER ::=
          { iso(1) member-body(2) 392 200011 61 security(1)
            algorithm(1) key-wrap-algorithm(3)
            camellia128-wrap(2) }

  id-camellia192-wrap OBJECT IDENTIFIER ::=
          { iso(1) member-body(2) 392 200011 61 security(1)
             algorithm(1) key-wrap-algorithm(3)
             camellia192-wrap(3) }

  id-camellia256-wrap OBJECT IDENTIFIER ::=
          { iso(1) member-body(2) 392 200011 61 security(1)
            algorithm(1) key-wrap-algorithm(3)
            camellia256-wrap(4) }
```

In all cases the parameters field of AlgorithmIdentifier MUST be ABSENT, because the key wrapping procedure itself defines how and when to use an IV. The OID gives the KEK key size, but does not make any statements as to the size of the wrapped Camellia CEK. Implementations MAY use different KEK and CEK sizes. Implementations MUST support the CEK and the KEK having the

same length. If different lengths are supported, the KEK MUST be
of equal or greater length than the CEK.

3. Key Wrap Algorithm

Camellia key wrapping and unwrapping is done in conformance with the AES key wrap algorithm [AES-WRAP][RFC3394], because Camellia and AES have the same block and key sizes, i.e. the block size of 128 bits and key sizes of 128, 192, and 256 bits.

3.1 Notation and Definitions

The following notation is used in the description of the key wrapping algorithms:

```
  Camellia(K, W)
                 Encrypt W using the Camellia codebook with key K
  Camellia-1(K, W)
                 Decrypt W using the Camellia codebook with key K
  MSB(j, W)      Return the most significant j bits of W
  LSB(j, W)      Return the least significant j bits of W
  B1 ^ B2        The bitwise exclusive or (XOR) of B1 and B2
  B1 | B2        Concatenate B1 and B2
  K              The key-encryption key K
  n              The number of 64-bit key data blocks
  s              The number of steps in the wrapping process, s = 6n
  P[i]           The ith plaintext key data block
  C[i]           The ith ciphertext data block
  A              The 64-bit integrity check register
  R[i]           An array of 64-bit registers where
                    i = 0, 1, 2, ..., n
  A[t], R[i][t] The contents of registers A and R[i] after encryption
                    step t.
  IV             The 64-bit initial value used during the wrapping
                    process.
```

In the key wrap algorithm, the concatenation function will be used to concatenate 64-bit quantities to form the 128-bit input to the Camellia codebook. The extraction functions will be used to split the 128-bit output from the Camellia codebook into two 64-bit quantities.

3.2 Camellia Key Wrap

Key wrapping with Camellia is identical to Section 2.2.1 of [RFC3394] with "AES" replaced by "Camellia".

The inputs to the key wrapping process are the KEK and the plaintext to be wrapped. The plaintext consists of n 64-bit blocks, containing the key data being wrapped. The key wrapping process is described below.

```
Inputs:      Plaintext, n 64-bit values {P1, P2, ..., Pn}, and
             Key, K (the KEK).
```

```
   Outputs:     Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.

   1) Initialize variables.
```

```
       Set A0 to an initial value (see Section 3.4)
       For i = 1 to n
            R[0][i] = P[i]

   2) Calculate intermediate values.

       For t = 1 to s, where s = 6n
           A[t] = MSB(64, Camellia(K, A[t-1] | R[t-1][1])) ^ t
           For i = 1 to n-1
               R[t][i] = R[t-1][i+1]
           R[t][n] = LSB(64, Camellia(K, A[t-1] | R[t-1][1]))

   3) Output the results.

       Set C[0] = A[t]
       For i = 1 to n
           C[i] = R[t][i]
```

An alternative description of the key wrap algorithm involves indexing rather than shifting. This approach allows one to calculate the wrapped key in place, avoiding the rotation in the previous description. This produces identical results and is more easily implemented in software.

```
   Inputs:  Plaintext, n 64-bit values {P1, P2, ..., Pn}, and
            Key, K (the KEK).
   Outputs: Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.

   1) Initialize variables.

       Set A = IV, an initial value (see Section 3.4)
       For i = 1 to n
           R[i] = P[i]

   2) Calculate intermediate values.

       For j = 0 to 5
           For i=1 to n
               B = Camellia(K, A | R[i])
               A = MSB(64, B) ^ t where t = (n*j)+i
               R[i] = LSB(64, B)

   3) Output the results.

       Set C[0] = A
       For i = 1 to n
           C[i] = R[i]
```

3.3 Camellia Key Unwrap

Key unwrapping with Camellia is identical to Section 2.2.2 of [RFC3394], with "AES" replaced by "Camellia".

The inputs to the unwrap process are the KEK and (n+1) 64-bit blocks of ciphertext consisting of previously wrapped key. It returns n blocks of plaintext consisting of the n 64-bit blocks of the decrypted key data.

```
Inputs:  Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and
         Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P1, P2, ..., Pn}.

1) Initialize variables.

    Set A[s] = C[0] where s = 6n
    For i = 1 to n
        R[s][i] = C[i]

2) Calculate the intermediate values.

    For t = s to 1
        A[t-1] = MSB(64, Camellia-1(K, ((A[t] ^ t) | R[t][n]))
        R[t-1][1] = LSB(64, Camellia-1(K, ((A[t]^t) | R[t][n]))
        For i = 2 to n
            R[t-1][i] = R[t][i-1]

3) Output the results.

    If A[0] is an appropriate initial value (see Section 3.4),
    Then
        For i = 1 to n
            P[i] = R[0][i]
    Else
        Return an error
```

The unwrap algorithm can also be specified as an index based operation, allowing the calculations to be carried out in place. Again, this produces the same results as the register shifting approach.

```
Inputs:  Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and
         Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P0, P1, K, Pn}.

1) Initialize variables.

    Set A = C[0]
    For i = 1 to n
        R[i] = C[i]

2) Compute intermediate values.
```

```
       For j = 5 to 0
           For i = n to 1
               B = Camellia-1(K, (A ^ t) | R[i]) where t = n*j+i
               A = MSB(64, B)
               R[i] = LSB(64, B)

   3) Output results.

   If A is an appropriate initial value (see Section 3.4),
   Then
       For i = 1 to n
           P[i] = R[i]
   Else
       Return an error
```

3.4 Key Data Integrity -- the Initial Value

The initial value (IV) refers to the value assigned to A[0] in the first step of the wrapping process. This value is used to obtain an integrity check on the key data. In the final step of the unwrapping process, the recovered value of A[0] is compared to the expected value of A[0]. If there is a match, the key is accepted as valid, and the unwrapping algorithm returns it. If there is not a match, then the key is rejected, and the unwrapping algorithm returns an error.

The exact properties achieved by this integrity check depend on the definition of the initial value. Different applications may call for somewhat different properties; for example, whether there is need to determine the integrity of key data throughout its lifecycle or just when it is unwrapped. This specification defines a default initial value that supports integrity of the key data during the period it is wrapped (in Section 3.4.1). Provision is also made to support alternative initial values (in Section 3.4.2).

3.4.1 Default Initial Value

The default initial value (IV) is defined to be the hexadecimal constant:

```
   A[0] = IV = A6A6A6A6A6A6A6A6
```

The use of a constant as the IV supports a strong integrity check on the key data during the period that it is wrapped. If unwrapping produces A[0] = A6A6A6A6A6A6A6A6, then the chance that the key data is corrupt is 2^-64. If unwrapping produces A[0] any other value, then the unwrap must return an error and not return any key data.

3.4.2 Alternative Initial Values

When the key wrap is used as part of a larger key management protocol or system, the desired scope for data integrity may be more than just the key data or the desired duration for more than just

   the period that it is wrapped. Also, if the key data is not just an
   Camellia key, it may not always be a multiple of 64 bits.
   Alternative definitions of the initial value can be used to address
   such problems. According to [RFC3394], NIST will define alternative

initial values in future key management publications as needed. In order to accommodate a set of alternatives that may evolve over time, key wrap implementations that are not application-specific will require some flexibility in the way that the initial value is set and tested.

4. SMIMECapabilities Attribute

An S/MIME client SHOULD announce the set of cryptographic functions it supports by using the S/MIME capabilities attribute. This attribute provides a partial list of OIDs of cryptographic functions and MUST be signed by the client. The functions' OIDs SHOULD be logically separated in functional categories and MUST be ordered with respect to their preference.

RFC 2633 [RFC2633], Section 2.5.2 defines the SMIMECapabilities signed attribute (defined as a SEQUENCE of SMIMECapability SEQUENCEs) to be used to specify a partial list of algorithms that the software announcing the SMIMECapabilities can support.

If an S/MIME client is required to support symmetric encryption with Camellia, the capabilities attribute MUST contain the Camellia OID specified above in the category of symmetric algorithms. The parameter associated with this OID MUST be CamelliaSMimeCapability.

```
   CamelliaSMimeCapabilty ::= NULL
```

The SMIMECapability SEQUENCE representing Camellia MUST be DER-encoded as the following hexadecimal strings:

Key Size	Capability
128	30 0f 06 0b 2a 83 08 8c 9a 4b 3d 01 01 01 02 05 00
196	30 0f 06 0b 2a 83 08 8c 9a 4b 3d 01 01 01 03 05 00
256	30 0f 06 0b 2a 83 08 8c 9a 4b 3d 01 01 01 04 05 00

When a sending agent creates an encrypted message, it has to decide which type of encryption algorithm to use. In general the decision process involves information obtained from the capabilities lists included in messages received from the recipient, as well as other information such as private agreements, user preferences, legal restrictions, and so on. If users require Camellia for symmetric encryption, it MUST be supported by the S/MIME clients on both the sending and receiving side, and it MUST be set in the user preferences.

5. Security Considerations

This document specifies the use of Camellia for encrypting the content of a CMS message and for encrypting the symmetric key used to encrypt the content of a CMS message, and the other mechanisms

are the same as the existing ones. Therefore, the security
considerations described in the CMS specifications [CMS][CMSALG] and
the AES key wrap algorithm [AES-WRAP][RFC3394] can be applied to
this document. No security problem has been found on Camellia

[CRYPTREC][NESSIE].

6. Intellectual Property Statement

7. Full Copyright Statement

8. References

[DES]   National Institute of Standards and Technology.
FIPS Pub 46: Data Encryption Standard. 15 January 1977.

[AES-WRAP] National Institute of Standards and Technology. AES Key
Wrap Specification. 17 November 2001.
http://csrc.nist.gov/encryption/kms/key-wrap.pdf

[CamelliaID] J. Nakajima and S. Moriai, "A Description of the
Camellia Encryption Algorithm", Internet-Draft, July 2001.
draft-nakajima-camellia-02.txt

[CamelliaSpec] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai,
J. Nakajima, and T. Tokita "Specification of Camellia - a
128-bit Block Cipher". http://info.isl.ntt.co.jp/camellia/

[CamelliaTech] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai,
J. Nakajima, and T. Tokita "Camellia: A 128-Bit Block Cipher
Suitable for Multiple Platforms - Design and Analysis -", In
Selected Areas in Cryptography, 7th Annual International
Workshop, SAC 2000, August 2000, Proceedings, Lecture Notes in
Computer Science 2012, pp.39--56, Springer-Verlag, 2001.

[CMS] R. Housley, "Cryptographic Message Syntax", RFC 3369, August
2002.

[CMSALG] R. Housley, "Cryptographic Message Syntax (CMS)
Algorithms", RFC 3370, August 2002.

[CRYPTREC] Information-technology Promotion Agency (IPA), Japan,
CRYPTREC. http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

[NESSIE] New European Schemes for Signatures, Integrity and
Encryption (NESSIE) project. http://www.cryptonessie.org

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

[RFC2633] Ramsdell, B., Editor. S/MIME Version 3 Message
Specification. RFC 2633. June 1999.

[RFC3394] J. Schaad and R. Housley, "Advanced Encryption Standard
(AES) Key Wrap Algorithm", RFC 3394, September 2002.

8. Authors' Address

Shiho Moriai
Sony Computer Entertainment Inc.

```
   Phone: +81-3-6438-7523
   FAX:   +81-3-6438-8629
   Email: camellia@isl.ntt.co.jp (Camellia team)
          shiho@rd.scei.sony.co.jp (Shiho Moriai)
```

Akihiro Kato
NTT Software Corporation
Phone: +81-45-212-7404
FAX: +81-45-212-7410
Email: akato@po.ntts.co.jp

Appendix A ASN.1 Module

```
CamelliaEncryptionAlgorithmInCMS
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs9(9) smime(16) modules(0) id-mod-cms-camellia(23) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- Camellia using CBC-chaining mode for key sizes of 128, 192, 256

id-camellia128-cbc OBJECT IDENTIFIER ::=
    { iso(1) member-body(2) 392 200011 61 security(1)
      algorithm(1) symmetric-encryption-algorithm(1)
      camellia128-cbc(2) }

id-camellia192-cbc OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) 392 200011 61 security(1)
     algorithm(1) symmetric-encryption-algorithm(1)
     camellia192-cbc(3) }

id-camellia256-cbc OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) 392 200011 61 security(1)
     algorithm(1) symmetric-encryption-algorithm(1)
     camellia256-cbc(4) }

-- Camellia-IV is a the parameter for all the above object identifiers.

Camellia-IV ::= OCTET STRING (SIZE(16))

-- Camellia S/MIME Capabilty parameter for all the above object
-- identifiers.

CamelliaSMimeCapability ::= NULL

-- Camellia Key Wrap Algorithm identifiers - Parameter is absent.

id-camellia128-wrap OBJECT IDENTIFIER ::=
    { iso(1) member-body(2) 392 200011 61 security(1)
      algorithm(1) key-wrap-algorithm(3)
      camellia128-wrap(2) }

id-camellia192-wrap OBJECT IDENTIFIER ::=
    { iso(1) member-body(2) 392 200011 61 security(1)
      algorithm(1) key-wrap-algorithm(3)
      camellia192-wrap(3) }
```

```
id-camellia256-wrap OBJECT IDENTIFIER ::=
    { iso(1) member-body(2) 392 200011 61 security(1)
      algorithm(1) key-wrap-algorithm(3)
      camellia256-wrap(4) }


END
```