Network Working Group
Request for Comments: 5649
Category: Informational

R. Housley
Vigil Security
M. Dworkin
NIST

August 2009

Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm

Abstract

This document specifies a padding convention for use with the AES Key Wrap algorithm specified in RFC 3394. This convention eliminates the requirement that the length of the key to be wrapped be a multiple of 64 bits, allowing a key of any practical length to be wrapped.

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

1. Introduction

Management of cryptographic keys often leads to situations where one symmetric key is used to encrypt and integrity-protect another key, which can be either a symmetric key or an asymmetric key. The operation is often called key wrapping.

This document specifies an extension of the Advanced Encryption Standard (AES) Key Wrap algorithm [AES-KW1, AES-KW2]. Without this extension, the input to the AES Key Wrap algorithm, called the key data, must be a sequence of two or more 64-bit blocks.

The AES Key Wrap with Padding algorithm can be used to wrap a key of any practical size with an AES key. The AES key-encryption key (KEK) must be 128, 192, or 256 bits. The input key data may be as short as one octet, which will result in an output of two 64-bit blocks (or 16 octets). Although the AES Key Wrap algorithm does not place a maximum bound on the size of the key data that can be wrapped, this extension does so. The use of a 32-bit fixed field to carry the octet length of the key data bounds the size of the input at 2^32 octets. Most systems will have other factors that limit the practical size of key data to much less than 2^32 octets.

A message length indicator (MLI) is defined as part of an "Alternative Initial Value" in keeping with the statement in Section 2.2.3.2 of [AES-KW1], which says:

> Also, if the key data is not just an AES key, it may not always be a multiple of 64 bits. Alternative definitions of the initial value can be used to address such problems.

2. Notation and Definitions

The following notation is used in the algorithm descriptions:

MSB(j, W)	Return the most significant j bits of W
LSB(j, W)	Return the least significant j bits of W
ENC(K, B)	AES Encrypt the 128-bit block B using key K
DEC(K, B)	AES Decrypt the 128-bit block B using key K
V1 | V2	Concatenate V1 and V2
K	The key-encryption key
m	The number of octets in the key data
n	The number of 64-bit blocks in the padded key data
Q[i]	The ith plaintext octet in the key data
P[i]	The ith 64-bit plaintext block in the padded key data
C[i]	The ith 64-bit ciphertext data block
A	The 64-bit integrity check register

3. Alternative Initial Value

The Alternative Initial Value (AIV) required by this specification is a 32-bit constant concatenated to a 32-bit MLI. The constant is (in hexadecimal) A65959A6 and occupies the high-order half of the AIV. Note that this differs from the high order 32 bits of the default IV in Section 2.2.3.1 of [AES-KW1], so there is no ambiguity between the two. The 32-bit MLI, which occupies the low-order half of the AIV, is an unsigned binary integer equal to the octet length of the plaintext key data, in network order -- that is, with the most significant octet first. When the MLI is not a multiple of 8, the key data is padded on the right with the least number of octets sufficient to make the resulting octet length a multiple of 8. The value of each padding octet shall be 0 (eight binary zeros).

Notice that for a given number of 64-bit plaintext blocks, there are only eight values of MLI that can have that outcome. For example, the only MLI values that are valid with four 64-bit plaintext blocks are 32 (with no padding octets), 31 (with one padding octet), 30, 29, 28, 27, 26, and 25 (with seven padding octets). When the unwrapping process specified below yields n 64-bit blocks of output data and an AIV, the eight valid values for the MLI are 8*n, (8*n)-1, ..., and (8*n)-7. Therefore, integrity checking of the AIV, which is contained in a 64-bit register called A, requires the following steps:

1) Check that MSB(32,A) = A65959A6.

2) Check that 8*(n-1) < LSB(32,A) <= 8*n. If so, let MLI = LSB(32,A).

3) Let b = (8*n)-MLI, and then check that the rightmost b octets of the output data are zero.

If all three checks pass, then the AIV is valid. If any of the checks fail, then the AIV is invalid and the unwrapping operation must return an error.

4. Specification of the AES Key Wrap with Padding Algorithm

The AES Key Wrap with Padding algorithm consists of a wrapping process and an unwrapping process, both based on the AES codebook [AES]. It provides an extension to the AES Key Wrap algorithm [AES-KW1, AES-KW2] that eliminates the requirement that the length of the key to be wrapped be a multiple of 64 bits. The next two sections specify the wrapping and unwrapping processes, called the

extended key wrapping process and the extended key unwrapping process, respectively. These names distinguish these processes from the ones specified in [AES-KW1] and [AES-KW2].

4.1. Extended Key Wrapping Process

The inputs to the extended key wrapping process are the KEK and the plaintext to be wrapped. The plaintext consists of between one and 2^32 octets, containing the key data being wrapped. The key wrapping process is described below.

```
Inputs:  Plaintext, m octets {Q[1], Q[2], ..., Q[m]}, and
         Key, K (the KEK).
Outputs: Ciphertext, (n+1) 64-bit values {C[0], C[1], ..., C[n]}.
```

1) Append padding

If m is not a multiple of 8, pad the plaintext octet string on the right with octets {Q[m+1], ..., Q[r]} of zeros, where r is the smallest multiple of 8 that is greater than m. If m is a multiple of 8, then there is no padding, and r = m.

Set n = r/8, which is the same as CEILING(m/8).

```
For i = 1, ..., n
   j = 8*(i-1)
   P[i] = Q[j+1] | Q[j+2] | ... | Q[j+8].
```

2) Wrapping

If the padded plaintext contains exactly eight octets, then prepend the AIV as defined in Section 3 above to P[1] and encrypt the resulting 128-bit block using AES in ECB mode [Modes] with key K (the KEK). In this case, the output is two 64-bit blocks C[0] and C[1]:

```
C[0] | C[1] = ENC(K, A | P[1]).
```

Otherwise, apply the wrapping process specified in Section 2.2.1 of [AES-KW2] to the padded plaintext {P[1], ..., P[n]} with K (the KEK) and the AIV as defined in Section 3 above as the initial value. The result is n+1 64-bit blocks {C[0], C[1], ..., C[n]}.

4.2. Extended Key Unwrapping Process

The inputs to the extended key unwrapping process are the KEK and (n+1) 64-bit ciphertext blocks consisting of a previously wrapped key. If the ciphertext is a validly wrapped key, then the unwrapping process returns n 64-bit blocks of padded plaintext, which are then mapped in this extension to m octets of decrypted key data, as indicated by the MLI embedded in the AIV.

Inputs: Ciphertext, (n+1) 64-bit blocks {C[0], C[1], ..., C[n]}, and Key, K (the KEK).
Outputs: Plaintext, m octets {Q[1], Q[2], ..., Q[m]}, or an error.

1) Key unwrapping

When n is one (n=1), the ciphertext contains exactly two 64-bit blocks (C[0] and C[1]), and they are decrypted as a single AES block using AES in ECB mode [Modes] with K (the KEK) to recover the AIV and the padded plaintext key:

A | P[1] = DEC(K, C[0] | C[1]).

Otherwise, apply Steps 1 and 2 of the unwrapping process specified in Section 2.2.2 of [AES-KW2] to the n+1 64-bit ciphertext blocks, {C[0], C[1], ..., C[n]}, and to the KEK, K. Define the padded plaintext blocks, {P[1], ..., P[n]}, as specified in Step 3 of that process, with A[0] as the A value. Note that checking "If A[0] is an appropriate value" is slightly delayed to Step 2 below since the padded plaintext is needed to perform this verification when the AIV is used.

2) AIV verification

Perform the three checks described in Section 3 above on the padded plaintext and the A value. If any of the checks fail, then return an error.

3) Remove padding

Let m = the MLI value extracted from A.

Let P = P[1] | P[2] | ... | P[n].

For i = 1, ... , m
Q[i] = LSB(8, MSB(8*i, P))

5. Algorithm Identifiers

Some security protocols employ ASN.1 [X.680] and employ algorithm identifiers to name cryptographic algorithms. To support these protocols, the AES Key Wrap with Padding algorithm has been assigned the following algorithm identifiers, one for each AES KEK size. The AES Key Wrap (without padding) algorithm identifiers are also included here for convenience.

```
aes OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
          us(840) organization(1) gov(101) csor(3)
          nistAlgorithm(4) 1 }

id-aes128-wrap     OBJECT IDENTIFIER ::= { aes 5 }
id-aes128-wrap-pad OBJECT IDENTIFIER ::= { aes 8 }

id-aes192-wrap     OBJECT IDENTIFIER ::= { aes 25 }
id-aes192-wrap-pad OBJECT IDENTIFIER ::= { aes 28 }

id-aes256-wrap     OBJECT IDENTIFIER ::= { aes 45 }
id-aes256-wrap-pad OBJECT IDENTIFIER ::= { aes 48 }
```

In all cases, the AlgorithmIdentifier parameter field MUST be absent.

6. Padded Key Wrap Examples

The examples in this section were generated using the index-based implementation of the AES Key Wrap algorithm along with the padding approach specified in Section 4 of this document. All values are shown in hexadecimal.

The first example wraps 20 octets of key data with a 192-bit KEK.

```
KEK   :  5840df6e29b02af1 ab493b705bf16ea1 ae8338f4dcc176a8

Key   :  c37b7e6492584340 bed1220780894115 5068f738

Wrap  :  138bdeaa9b8fa7fc 61f97742e72248ee 5ae6ae5360d1ae6a
      :  5f54f373fa543b6a
```

The second example wraps 7 octets of key data with a 192-bit KEK.

```
KEK   :  5840df6e29b02af1 ab493b705bf16ea1 ae8338f4dcc176a8

Key   :  466f7250617369

Wrap  :  afbeb0f07dfbf541 9200f2ccb50bb24f
```

7. Security Considerations

Implementations must protect the key-encryption key (KEK). Compromise of the KEK may result in the disclosure of all keys that have been wrapped with the KEK, which may lead to the compromise of all traffic protected with those wrapped keys.

The KEK must be at least as good as the keying material it is protecting.

If the KEK and wrapped key are associated with different cryptographic algorithms, the effective security provided to data protected with the wrapped key is determined by the weaker of the two algorithms. If, for example, data is encrypted with 128-bit AES and that AES key is wrapped with a 256-bit AES key, then at most 128 bits of protection is provided to the data. If, for another example, a 128-bit AES key is used to wrap a 4096-bit RSA private key, then at most 128 bits of protection is provided to any data that depends on that private key. Thus, implementers must ensure that key-encryption algorithms are at least as strong as other cryptographic algorithms employed in an overall system.

The AES Key Wrap and the AES Key Wrap with Padding algorithms use different constants in the initial value. The use of different values ensures that the recipient of padded key data cannot successfully unwrap it as unpadded key data, or vice versa. This remains true when the key data is wrapped using the AES Key Wrap with Padding algorithm but no padding is needed.

The AES Key Wrap with Padding algorithm provides almost the same amount of integrity protection as the AES Key Wrap algorithm.

A previous padding technique was specified for wrapping Hashed Message Authentication Code (HMAC) keys with AES [OLD-KW]. The technique in this document is more general; the technique in this document is not limited to wrapping HMAC keys.

In the design of some high assurance cryptographic modules, it is desirable to segregate cryptographic keying material from other data. The use of a specific cryptographic mechanism solely for the protection of cryptographic keying material can assist in this goal. The AES Key Wrap and the AES Key Wrap with Padding are such mechanisms. System designers should not use these algorithms to encrypt anything other than cryptographic keying material.

8. References

8.1. Normative References

[AES] National Institute of Standards and Technology, FIPS Pub 197: Advanced Encryption Standard (AES), 26 November 2001.

[AES-KW1] National Institute of Standards and Technology, AES Key Wrap Specification, 17 November 2001. http://csrc.nist.gov/groups/ST/toolkit/documents/kms/AES_key_wrap.pdf

[AES-KW2] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, September 2002.

[Modes] Dworkin, M., "Recommendation for Block Cipher Modes of Operation -- Methods and Techniques", NIST Special Publication 800-38A, 2001.

[X.680] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

8.2. Informative References

[AES-CMS] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

[CMS-ASN] Schaad, J. and P. Hoffman, "New ASN.1 Modules for CMS and S/MIME", Work in Progress, August 2009.

[OLD-KW] Schaad, J. and R. Housley, "Wrapping a Hashed Message Authentication Code (HMAC) key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key", RFC 3537, May 2003.

[X.681] ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002, Information Technology - Abstract Syntax Notation One: Information Object Specification.

[X.682] ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002, Information Technology - Abstract Syntax Notation One: Constraint Specification.

[X.683] ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002, Information Technology - Abstract Syntax Notation One: Parameterization of ASN.1 Specifications.

9. Acknowledgments

Paul Timmel should be credited with the MLI and padding technique described in this document.

Appendix A. ASN.1 Modules

This appendix includes two ASN.1 modules. The first one makes use of the 1988 syntax, and the second one makes use of the 2002 ASN.1 syntax.

Appendix A.1 provides the normative ASN.1 definitions for the algorithm identifiers included in this specification using ASN.1 as defined in [X.680] using the 1988 ASN.1 syntax.

Appendix A.2 provides informative ASN.1 definitions for the algorithm identifiers included in this specification using ASN.1 as defined in [X.680], [X.681], [X.682], and [X.683] using the 2002 ASN.1 syntax. This appendix contains the same information as Appendix A.1; however, Appendix A.1 takes precedence in case of conflict. The content encryption and key wrap algorithm objects are defined in [CMS-ASN].

The id-aes128-wrap, id-aes192-wrap, and id-aes256-wrap algorithm identifiers are defined in [AES-CMS].

A.1. 1988 ASN.1 Module

```
AESKeyWrapWithPad-88 { iso(1) member-body(2) us(840) rsadsi(113549)
  pkcs(1) pkcs-9(9) smime(16) modules(0) 47 }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

-- IMPORTS NONE --

-- AES information object identifiers --

aes OBJECT IDENTIFIER ::= {
  joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101)
  csor(3) nistAlgorithms(4) 1 }

-- AES Key Wrap With Padding Algorithm Identifiers are to be used
-- with the Parameter field absent

id-aes128-wrap-pad OBJECT IDENTIFIER ::= { aes 8 }
id-aes192-wrap-pad OBJECT IDENTIFIER ::= { aes 28 }
id-aes256-wrap-pad OBJECT IDENTIFIER ::= { aes 48 }

END
```

A.2. 2002 ASN.1 Module

```
AESKeyWrapWithPad-02 { iso(1) member-body(2) us(840) rsadsi(113549)
  pkcs(1) pkcs-9(9) smime(16) modules(0) 48 }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS
  AlgorithmIdentifier{}, CONTENT-ENCRYPTION, KEY-WRAP, SMIME-CAPS
  FROM AlgorithmInformation-2009  -- [CMS-ASN]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithmInformation-02(58) };

AES-ContentEncryption CONTENT-ENCRYPTION ::= {
  cea-aes128-wrap-pad |
  cea-aes192-wrap-pad |
  cea-aes256-wrap-pad,
  ... }

AES-KeyWrap KEY-WRAP ::= {
  kwa-aes128-wrap-pad |
  kwa-aes192-wrap-pad |
  kwa-aes256-wrap-pad,
  ... }

SMimeCaps SMIME-CAPS ::= {
  cea-aes128-wrap-pad.&smimeCaps |
  cea-aes192-wrap-pad.&smimeCaps |
  cea-aes256-wrap-pad.&smimeCaps |
  kwa-aes128-wrap-pad.&smimeCaps |
  kwa-aes192-wrap-pad.&smimeCaps |
  kwa-aes256-wrap-pad.&smimeCaps,
  ... }

-- AES object identifier

aes OBJECT IDENTIFIER ::= {
  joint-iso-itu-t(2) country(16) us(840) organization(1)
  gov(101) csor(3) nistAlgorithms(4) 1 }
```

```
   -- Content Encryption Algorithms

   cea-aes128-wrap-pad CONTENT-ENCRYPTION ::= {
     IDENTIFIER id-aes128-wrap-pad
     PARAMS ARE absent
     SMIME-CAPS { IDENTIFIED BY id-aes128-wrap-pad } }

   cea-aes192-wrap-pad CONTENT-ENCRYPTION ::= {
     IDENTIFIER id-aes192-wrap-pad
     PARAMS ARE absent
     SMIME-CAPS { IDENTIFIED BY id-aes192-wrap-pad } }

   cea-aes256-wrap-pad CONTENT-ENCRYPTION ::= {
     IDENTIFIER id-aes256-wrap-pad
     PARAMS ARE absent
     SMIME-CAPS { IDENTIFIED BY id-aes256-wrap-pad } }

   -- Key Wrap Algorithms

   kwa-aes128-wrap-pad KEY-WRAP ::= {
     IDENTIFIER id-aes128-wrap-pad
     PARAMS ARE absent
     SMIME-CAPS { IDENTIFIED BY id-aes128-wrap-pad } }

   id-aes128-wrap-pad OBJECT IDENTIFIER ::= { aes 8 }

   kwa-aes192-wrap-pad KEY-WRAP ::= {
     IDENTIFIER id-aes192-wrap-pad
     PARAMS ARE absent
     SMIME-CAPS { IDENTIFIED BY id-aes192-wrap-pad } }

   id-aes192-wrap-pad OBJECT IDENTIFIER ::= { aes 28 }

   kwa-aes256-wrap-pad KEY-WRAP ::= {
     IDENTIFIER id-aes256-wrap-pad
     PARAMS ARE absent
     SMIME-CAPS { IDENTIFIED BY id-aes256-wrap-pad } }

   id-aes256-wrap-pad OBJECT IDENTIFIER ::= { aes 48 }

   END
```

Authors' Addresses

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA

EMail: housley@vigilsec.com

Morris Dworkin
National Institute of Standards and Technology
100 Bureau Drive, Mail Stop 8930
Gaithersburg, MD 20899-8930
USA

EMail: dworkin@nist.gov