INTERNET DRAFT R. Housley
Intended Status: Informational Vigil Security
M. Dworkin
NIST
Expires: 24 September 2009 24 March 2009

Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm
<draft-housley-aes-key-wrap-with-pad-02.txt>

Status of this Memo

Copyright Notice

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 is a multiple of 64 bits, allowing a key of any practical length to be wrapped.

1. Introduction

Management of cryptographic keys often leads to situations where a 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 9 octets, 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 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
B1 | B2	Concatenate B1 and B2
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 [AES-KW1] Section 2.2.3.1, 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 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 AES Key Unwrap yields n 64-bit blocks of key data with an AIV, the eight valid values for the MLI are 8*n, (8*n)-1, ..., and (8*n)-7. Therefore, the integrity check for the AIV 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 plaintext 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 AES Key Unwrap 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 is 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,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 9 and 2^32 octets, containing the key data being wrapped. The key wrapping process is described below.

```
Inputs:      Plaintext, m octets {Q1, Q2, ..., Qm}, and
             Key, K (the KEK).
Outputs:     Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.
```

1) Append padding

If m is not a multiple of 8, pad the plaintext octet string on the right with octets {Qm+1, ..., Qr} 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

Apply the wrapping process specified in Section 2.2.1 of [AES-KW2] to the padded plaintext {P1, ..., Pn} and K (the KEK), with the AIV as defined in Section 3 above as the initial value.

The result is n+1 64-bit blocks {C0, C1, ..., Cn}.

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 (original) unwrapping process returns n 64-bit plaintext blocks, 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 {C0, C1, ..., Cn}, and Key, K (the KEK).
Outputs:	Plaintext, m octets {Q1, Q2, ..., Qm}, or an error.

1) Key unwrapping

Apply Steps 1 and 2 of the unwrapping process specified in Section 2.2.2 of [AESKW2] to the n+1 64-bit ciphertext blocks, {C0, C1, ..., Cn}, and the KEK, K. Define the padded plaintext blocks, {P1, ..., Pn}, 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 = P1 | P2 | ... | Pn.

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

5. Algorithm Identifiers

Some security protocols employ ASN.1 [X.690], and these protocols 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 Example

The example in this section was generated using the index-based implementation of the AES Key Wrap algorithm along with the padding approach specified in Section 4 of this document. The example wraps 20 octets of Key Data with a 192-bit KEK. All values are shown in hexadecimal.

```
      KEK    :  5840df6e29b02af1 ab493b705bf16ea1 ae8338f4dcc176a8
```

```
Key    :  c37b7e6492584340 bed1220780894115 5068f738

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

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.

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 as strong or stronger than 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 HMAC keys with AES [OLD-KW]. The technique in this document is preferred, and the technique in this document is not limited to wrapping HMAC keys.

The key wrapping technique specified in this document requires the length of the key data to be at least nine octets because a single application of the AES codebook is sufficient to protect up to eight octets of key data. In particular, if the key data consists of eight

or fewer octets, then a 64-bit integrity check value could be prepended to the key data to form a single 128-bit block. For example, the integrity check value could consist of a fixed seven octet value followed by a single octet length value. The wrapping and unwrapping processes employing such an integrity check value and a single AES codebook operation could be defined analogous to those in Section 4 if there is a need to wrap keys that are smaller than nine octets.

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/encryption/kms/key-wrap.pdf]

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

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

OLD-KW J. Schaad 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.

9. Acknowledgments

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

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