

IEEE P1619.2 D?

<put date here>

Draft Standard Architecture for Wide-Block Encryption for Shared Storage Media

**Sponsored by the
Security in Storage Workgroup
of the IEEE Computer Society**

Copyright © 2008
by the Institute of Electrical and Electronics Engineers, Inc.
Three Park Avenue
New York, New York 10016-5997, USA
All rights reserved.

This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to change. **USE AT YOUR OWN RISK!** Because this is an unapproved draft, this document must not be utilized for any conformance/compliance purposes. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only. Prior to submitting this document to another standards development organization for standardization activities, permission must first be obtained from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. Other entities seeking permission to reproduce this document, in whole or in part, must obtain permission from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department.

IEEE Standards Activities Department
Standards Licensing and Contracts
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

Introduction

The purpose of IEEE1619.2 standard is ...

At the time this standard was completed, the working group had the following membership:

The following members of the balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention. (To be provided by IEEE editor at time of publication.)

<put date here>

IEEE P1619.2/D?

Contents

Wide-Block Encryption for Shared Storage Media

1. Overview

1.1 Scope and Purpose

This standard specifies an architecture for encryption of data in random access storage devices, oriented toward applications which benefit from wide encryption-block sizes of 512 bytes and above.

1.2 Related work

This standard specifies an architecture for media security and enabling components. Wide encryption blocks are well suited to environments where the attacker has repeated access to cryptographic communication or ciphertext, or is able to perform traffic analysis of data access patterns. The standard is oriented toward fixed-size encryption blocks without data expansion, but anticipates an optional data expansion mode to resist attacks involving data tampering.

2. References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

NIST FIPS-197, Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard.¹

3. Definitions

None yet.

4. The XCB-AES Transform

Insert here the XCB draft.

5. The EME2-AES Transform

5.1 The Mult-by-alpha Operation

To be written.

¹ FIPS publications are available from the National Technical Information Service (NTIS), 5285 Port Royal Road, Springfield, VA, USA. FIPS-197 is also available on-line from <http://csrc.nist.gov/CryptoToolkit/aes/>

5.1 EME2-AES Encryption

The EME2-AES encryption procedure can be described by the formula:

$$C = \text{EME2-AES-Enc}(Key, T, P),$$

where

Key is the 384 or 512 bit EME2-AES key

T is a tweak value, of arbitrary bit length (zero or more bits)

P is the plaintext, of length 128 bits or more

C is the ciphertext resulting from the operation, of the same bit-length as *P*

The input to the EME2-AES encryption routine is parsed as follows:

- The key is partitioned into three fields, $Key = Key_1 \mid Key_2 \mid Key_3$, with Key_3 consisting of the last 128 bits, Key_2 consisting of the 128 bits before them, and Key_1 consisting of the remaining first 128 or 256 bits.
- If not empty, the tweak is partitioned into a sequence of blocks $T = T_1 \mid T_2 \mid \dots \mid T_r$, where each of the blocks T_1, T_2, \dots, T_{r-1} is of length exactly 128 bits, and T_r is of length between 1 and 128 bits.
- The plaintext *P* is partitioned into a sequence of blocks $P = P_1 \mid P_2 \mid \dots \mid P_m$, where each of the blocks P_1, P_2, \dots, P_{m-1} is of length exactly 128 bits, and P_m is of length between 1 and 128 bits.

The ciphertext shall then be computed by the sequence of steps in Figure 2 or equivalent. An illustration of these steps (for plaintext of 129 full blocks and one partial block) is provided in Figure 1.

Figure 1. An illustration of EME2-AES

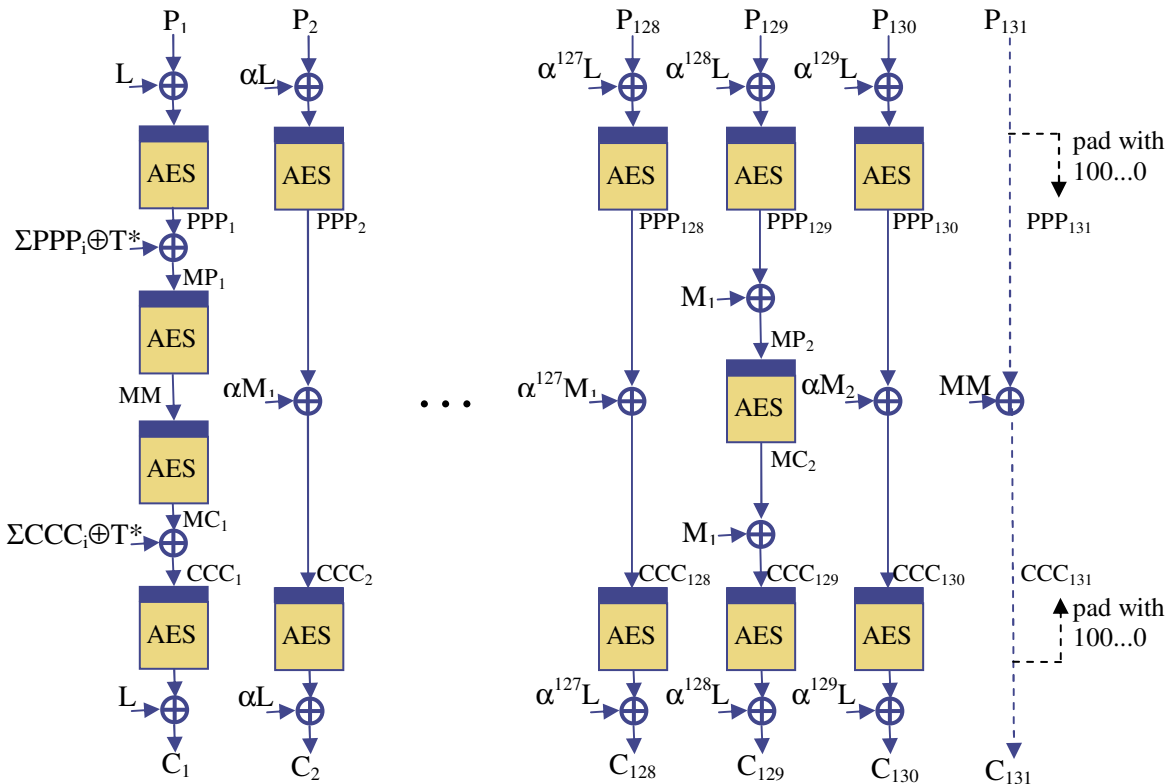


Figure 2. The EME2-AES Encryption Procedure

```

// Process the tweak T to get the 128-bit block T_star
1. if len(T)==0 then T_star = AES-Enc(Key1, Key3)
2. else
3.   Key3 = Mult-by-alpha(Key3)
4.   for i=1 to r-1
5.     TTi = AES-Enc(Key1, Key3⊕Ti) ⊕ Key3
6.     Key3 = Mult-by-alpha(Key3)
7.   If len(Tr)<128 then
8.     Tr = Tr | 10...0 // pad Tr to 128 bits
9.     Key3 = Mult-by-alpha(Key3)
10.  TTr = AES-Enc(Key1, Key3⊕Tr) ⊕ Key3
11.  T_star = TT1 ⊕ TT2 ⊕ ... ⊕ TTr

// First ECB pass
12. L = Key2
13. for i=1 to m-1
14.  PPPi = AES-Enc(Key1, L⊕Pi)
15.  L = Mult-by-alpha(L)
16. if len(Pm)<128 then PPPm = Pm | 10...0 // pad Pm to 128 bits
17. else
18.   PPPm = AES-Enc(Key1, L⊕Pm)

// Intermeidate mixing
18. MP = PPP1 ⊕ PPP2 ⊕ ... ⊕ PPPm ⊕ T_star
19. if len(Pm)<128 then
20.  MM = AES-Enc(Key1, MP)
21.  MC = MC1 = AES-Enc(Key1, MM)
22. else MC = MC1 = AES-Enc(Key1, MP)
23. M = M1 = MP ⊕ MC
24. for i=2 to m-1
25.  if (i-1 mod 128 > 0) then
26.    M = Mult-by-alpha(M)
27.    CCCi = PPPi ⊕ M
28.  else
29.    MP = PPPi ⊕ M1
30.    MC = AES-Enc(Key1, MP)
31.    M = MP ⊕ MC
32.    CCCi = MC ⊕ M1
33. if len(Pm)<128 then
34.  Cm = Pm ⊕ (MM truncated to len(Pm) bits)
35.  CCCm = Cm | 10...0 // pad Cm to 128 bits
36. else if (m-1 mod 128 > 0) then
37.  M = Mult-by-alpha(M)
38.  CCCm = PPPm ⊕ M
39. else CCCm = AES-Enc(Key1, M1⊕PPPm) ⊕ M1
40. CCC1 = MC1 ⊕ CCC2 ⊕ ... ⊕ CCCm ⊕ T_star

// Second ECB Pass
41. L = Key2
42. for i=1 to m-1
43.  Ci = AES-Enc(Key1, CCCi) ⊕ L
44.  L = Mult-by-alpha(L)
45. if len(Pm)==128 then Cm = AES-Enc(Key1, CCCm) ⊕ L

```

5.2 EME2-AES Decryption

The EME2-AES decryption procedure can be described by the formula:

$$C = \text{EME2-AES-Dec}(Key, T, C),$$

where

- Key* is the 256 or 384 bit EME2-AES key
- T* is a tweak value, of arbitrary bit length (zero or more bits)
- C* is the ciphertext, of length 128 bits or more
- P* is the plaintext resulting from the operation, of the same bit-length as *C*

The input to the EME2-AES decryption routine is parsed as follows:

- The key is partitioned into three fields, $Key = Key_1 \mid Key_2 \mid Key_3$, with Key_3 consisting of the last 128 bits, Key_2 consisting of the 128 bits before them, and Key_1 consisting of the remaining first 128 or 256 bits.
- If not empty, the tweak is partitioned into a sequence of blocks $T = T_1 \mid T_2 \mid \dots \mid T_r$, where each of the blocks T_1, T_2, \dots, T_{r-1} is of length exactly 128 bits, and T_r is of length between 1 and 128 bits.
- The ciphertext *P* is partitioned into a sequence of blocks $C = C_1 \mid C_2 \mid \dots \mid C_m$, where each of the blocks C_1, C_2, \dots, C_{m-1} is of length exactly 128 bits, and C_m is of length between 1 and 128 bits.

The ciphertext shall then be computed by the sequence of steps in Figure 3 or equivalent. (Note that the only difference between the encryption and decryption routines is that all the AES-Enc operations within lines 12-45 are replaced by AES-Dec operations.)

Figure 3. The EME2-AES Decryption routine

```

// Process the tweak T to get the 128-bit block T_star
1. if len(T)==0 then T_star = AES-Enc(Key1, Key3)
2. else
3.   Key3 = Mult-by-alpha(Key3)
4.   for i=1 to r-1
5.     TTi = AES-Enc(Key1, Key3⊕Ti) ⊕ Key3
6.     Key3 = Mult-by-alpha(Key3)
7.   If len(Tr)<128 then
8.     Tr = Tr | 10...0 // pad Tr to 128 bits
9.     Key3 = Mult-by-alpha(Key3)
10.  TTr = AES-Enc(Key1, Key3⊕Tr) ⊕ Key3
11.  T_star = TT1 ⊕ TT2 ⊕ ... ⊕ TTr

// First ECB pass
12. L = Key2
13. for i=1 to m-1
14.  CCCi = AES-Dec(Key1, L⊕Ci)
15.  L = Mult-by-alpha(L)
16. if len(Cm)<128 then CCCm = Cm | 10...0 // pad Cm to 128 bits
17. else CCCm = AES-Dec(Key1, L⊕Cm)

// Intermeidate mixing
18. MC = CCC1 ⊕ CCC2 ⊕ ... ⊕ CCCm ⊕ T_star
19. if len(Cm)<128 then
20.  MM = AES-Dec(Key1, MC)
21.  MP = MP1 = AES-Dec(Key1, MM)
22. else MP = MP1 = AES-Dec(Key1, MC)

```

```
23. M = M1 = MP ⊕ MC
24. for i=2 to m-1
25.   if (i-1 mod 128 > 0) then
26.     M = Mult-by-alpha(M)
27.     PPPi = CCCi ⊕ M
28.   else
29.     MC = CCCi ⊕ M1
30.     MP = AES-Dec(Key1, MC)
31.     M = MP ⊕ MC
32.     PPPi = MP ⊕ M1
33. if len(Cm) < 128 then
34.   Pm = Cm ⊕ (MM truncated to len(Cm) bits)
35.   PPPm = Pm | 10...0 // pad Pm to 128 bits
36. else if (m-1 mod 128 > 0) then
37.   M = Mult-by-alpha(M)
38.   PPPm = CCCm ⊕ M
39. else PPPm = AES-Dec(Key1, M1 ⊕ CCCm) ⊕ M1
40. PPP1 = MP1 ⊕ PPP2 ⊕ ... ⊕ PPPm ⊕ T_star

// Second ECB Pass
41. L = Key2
42. for i=1 to m-1
43.   Pi = AES-Dec(Key1, PPPi) ⊕ L
44.   L = Mult-by-alpha(L)
45. if len(Cm) == 128 then Pm = AES-Dec(Key1, PPPm) ⊕ L
```

Annex A: Bibliography (informative)