IEEE P1619.2™/D3

Draft Standard for Wide-Block Encryption for Shared Storage Media

Prepared by the Security in Storage Working Group Working Group of the IEEE Computer Society Information Assurance Committee

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 international standardization consideration. Prior to adoption of this document, in whole or in part, by another standards development organization, permission must first be obtained from the IEEE Standards Activities Department (stds.ipr@ieee.org). Other entities seeking permission to reproduce this document, in whole or in part, must also obtain permission from the IEEE Standards Activities Department.

IEEE Standards Activities Department
445 Hoes Lane
Piscataway, NJ 08854, USA

Copyright © 2008 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
Abstract: This standard specifies an architecture for encryption of data in random access storage devices, oriented towards applications which benefit from wide encryption-block sizes of 512 bytes and above.

Keywords: <Select this text and type or paste keywords>
This page is left blank intentionally.
Introduction

This introduction is not part of IEEE P<designation>/D<draft_number>, Draft Standard for Wide-Block Encryption for Shared Storage Media.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA web site at http://standards.ieee.org.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.
Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this Standard may require use of subject matter covered by patent rights. By publication of this Standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this Standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this draft Standard was completed, the 1619.2 Working Group had the following membership:

James Hughes, Chair

Serge Plotkin, Vice Chair

Fabio Maino, Editor

The following members of the [individual/entity] balloting committee voted on this Standard. Balloters may have voted for approval, disapproval, or abstention.

(part to be supplied by IEEE)
CONTENTS

<AFTER DRAFT BODY IS COMPLETE, SELECT THIS TEXT AND CLICK INSERT SPECIAL->ADD (TABLE OF) CONTENTS>

Deleted: 2
Deleted: May
Draft Standard for Wide-Block Encryption for Shared Storage Media

1. Overview

1.1 Scope

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 Purpose

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. Normative 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 and corrigenda) applies.

3. Definitions, Acronyms, and Abbreviations

3.1 Definitions

For the purposes of this draft standard, the following terms and definitions apply. The Authoritative Dictionary of IEEE Standards, Seventh Edition, should be referenced for terms not defined in this clause.
Encryption with associated data (EAD): a cryptographic method that consist of an encrypt and decrypt operation used to encrypt a plaintext and the associated data with a secret key, or to decrypt a ciphertext and the associate encrypted data with the same secret key. In other methods for encryption of data at rest, the associated data is often referred as “tweak”.

Tweak value: The 128-bit value sometimes used in other standards to represent the logical position of the data being encrypted or decrypted with a wide-block encryption mode. Tweakable encryption usually refers to fix length tweak. Across this document the term “Associated Data” is used to refer to the concept of tweak.

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>advanced encryption standard</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standard</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field (see Menezes et al. [??])</td>
</tr>
<tr>
<td>LBA</td>
<td>logical block address</td>
</tr>
<tr>
<td>GCM</td>
<td>Galois/Counter Mode of authenticated encryption</td>
</tr>
</tbody>
</table>

4. Mathematical Conventions

This standard uses decimal, binary, and hexadecimal numbers. For clarity, decimal numbers generally represent counts, and binary or hexadecimal numbers describe bit patterns or raw binary data.

Binary numbers are represented by a string of one or more binary digits, followed by the subscript 2. For example, the decimal number 26 is represented as 00011010 in binary.

Hexadecimal numbers are represented by a string of one or more hexadecimal digits, prefixed by the string “0x”. For example, the decimal number 135 is represented as 0x87 in hexadecimal.

The main functions used in this standard are the AES block cipher encryption, the multiplication-by-alpha operation, and the multiplication over the field GF($2^{128}$).

The AES block cipher encryption of the value $X$ with the key $K$ is denoted as AES-enc($K, X$), and the AES block cipher decryption is denoted as AES-dec($K, X$).

The multiplication-by-alpha operation of a 16-byte value $X \in GF(2^{128})$ by a primitive element $\alpha$ in the field GF($2^{128}$) is denoted as mult-by-alpha($X$) and is defined in Section 5.2.1.

The multiplication of two elements $X, Y \in GF(2^{128})$ is denoted as $X \cdot Y$, and the addition of $X$ and $Y$ is denoted as $X \oplus Y$. Addition in this field is equivalent to the bitwise exclusive-or operation, and the multiplication operation is defined in Section 5.3.3. We denote the number of bits in a byte string $X$ as $\#X$.

The function len($S$) returns a 64-bit string containing the nonnegative integer describing the number of bits in its argument $S$, with the least significant bit on the right. The expression $0^n$ denotes a string of $n$ zero bits, and $A/B$ denotes the concatenation of two bit strings $A$ and $B$. The function msb($S$) returns the initial $t$ bits of the string $S$. We consider bit strings to be indexed starting on the left, so that bit zero of $S$ is the...
leftmost bit. When \( S \) is a bit string and \( 0 \leq a < b \leq \#S \), we denote as \( S_{[a; b]} \) the length \( b - a + 1 \) substring of \( S \) consisting of bits \( a \) through \( b \) of \( S \). The symbol \( \{ \} \) denotes the bit string with zero length.

The function \( \text{incr}: \{0, 1\}^{128} \rightarrow \{0, 1\}^{128} \) is the increment operation that is used to generate successive counter values. This function treats the rightmost 32 bits of its argument as a nonnegative integer with the least significant bit on the right, incrementing this value modulo \( 2^{32} \). More formally, \( \text{incr}(X) = X_{[0; 95]} | (X_{[96; 127]} + 1 \mod 2^{32}) \), where we rely on the implicit conversion of bit strings to integers.

5. Wide-block Encryption Algorithms

5.1 Data Units and Associated Data

The purpose of this standard is to specify Encryption with Associated Data (EAD) methods that are suitable for the encryption of data at rest. An EAD method consists of an encryption operation and a decryption operation. The encryption operation accepts three inputs: a secret key, a plaintext, and the associated data. It returns a single ciphertext value. Each of these inputs is regarded as an octet string.

The secret key must be unpredictable to the adversary. Each EAD algorithm accepts a key of a fixed length, but different algorithms may have keys of different lengths.

The plaintext input contains the data to be encrypted. Within a particular key scope, plaintext data units may have different lengths. An EAD method defines the range of admissible plaintext lengths.

The associated data input contains data that is associated with the plaintext, but which does not need to be encrypted. The choice of data for this input is described in more detail below. Within a particular key scope, associated data unit may have different lengths.

The ciphertext returned by the encryption operation is the same length as the plaintext.

The decryption operation takes three inputs: a secret key, a ciphertext, and an associated data value. It returns a single plaintext value. These values are as defined above, but with roles of the ciphertext and plaintext reversed.

The decryption operation is the reverse of the encryption operation; more specifically, if the encryption of the plaintext \( P \) with the key \( K \) and the associated data \( A \) results in the ciphertext \( C \), then the decryption of \( C \) with the key \( K \) and the associated data \( A \) will result in the plaintext \( P \).

This value of associated data must be known at the time of encryption and the time of decryption, so it should contain only information that is available, in plaintext form, at the time of both operations.

The associated data input should characterize the plaintext, and it should be as fine-grained as possible. This is because whenever the same plaintext is encrypted two different times using the same key but with distinct associated data values, the result is two distinct ciphertext values. Thus the use of distinct associated data values hides the equality of the plaintexts from an attacker.

5.1.1 Security Goal

The security goal of an EAD is as follows. It is assumed that an attacker can request the encryptions of multiple plaintext/associated data pairs with an unknown key, and request the decryptions of multiple ciphertext/associated data pairs as well. These plaintexts, ciphertexts, and associated data values can be
adaptively chosen by the adversary. The goal of the EAD is that its output, under such an attack, cannot be distinguished from random by a computationally limited adversary.

The different inputs to the EAD can be understood this way. If the plaintext is fixed, and different inputs are provided to the associated data, then the EAD encryption algorithm acts as a pseudorandom function; that is, a computationally limited adversary cannot distinguish it from a function selected uniformly at random from the set of all functions. If the associated data is fixed, and different inputs are provided to the plaintext, then the EAD encryption algorithm acts as a pseudorandom permutation; that is, a computationally limited adversary cannot distinguish it from an invertible function selected uniformly at random from the set of all invertible functions.

5.1.2 Using EAD to protect an array of data block

An EAD can be used to protect an array of data blocks, such as those in a data-storage disk. In this application, the associated data input to the encryption and decryption operations should contain the logical index of block on which the operation is acting. When this information is included in the associated data, cases in which two distinct data blocks contain identical plaintext values will be hidden from an adversary.

If multiple disks are being protected with a single key, then the associated data input should contain both the logical index of the block and an additional distinguishing parameter that is unique to each of the disks. When this information is included in the associated data, cases in which two distinct data blocks on different disks contain identical plaintext values will be hidden from an adversary.

[Editor Note: Need to have an example with Figure showing encryption of a block device - Fabio will provide text. Check format of EAD.Graffle file doesn’t work with OmniGraffle v5.0]
5.2 The EME2 Transform

5.2.1 The Mult-by-alpha Operation

The encryption and decryption procedures described in the following sections use a function Mult-by-alpha(X), that multiplies a 16-byte value X by a primitive element \( \alpha \) in the field \( \text{GF}(2^{128}) \). The input value is first converted into a byte array \( X[i], i = 0,1,...,15 \), where \( X[0] \) is the first byte of the byte array.

The multiplication by alpha is defined by the following, or a mathematically equivalent, procedure:

```
Mult-by-alpha(X)

Input: byte array \( X[i], i = 0,1,...,15 \)
Output: byte array \( Y[i], i = 0,1,...,15 \)

for i=0 to 15 do
    \( Y[i] = 2 \times X[i] \mod 256 \)
    if (i>0 and \( X[i-1] > 127 \)) then \( Y[i] = Y[i]-1 \)
end-for

if (\( X[15] > 127 \)) then \( Y[0] = Y[0] \text{xor } 0x87 \)
```

NOTE - Conceptually, the operation is a left shift of each byte by one bit with carry propagating from one byte to the next. Also, if the 15\textsuperscript{th} (last) byte shift results in a carry, a special value (hexadecimal \( 0x87 \)) is xor'ed into the first byte. This value is derived from the modulus of the Galois Field (polynomial \( x^{128}+x^7+x^2+x+1 \)).

5.2.2 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 18 or 64 byte EME2-AES key
- \( T \) is the value of the associated data of arbitrary byte length (zero or more bytes)
- \( P \) is the plaintext, of length 16 bytes or more
- \( C \) is the ciphertext resulting from the operation, of the same byte-length as \( P \)

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

\[
\text{Comment: Even if the algo is now in byte, shouldn't we leave the key length expressed in bits?}
\text{Comment: I've removed the use of tweak through the doc}
\text{Comment: Key is the 384 or 512 bit EME2-AES key}
\text{Comment: T is a tweak value, of arbitrary bit length (zero or more bits)}
\text{Comment: P is the plaintext, of length 128 bits or more}
\text{Comment: C is the ciphertext resulting from the operation, of the same bit-length as P}
\text{Copyright © <year> IEEE. All rights reserved.}
\text{This is an unapproved IEEE Standards Draft, subject to change.
The key is partitioned into three fields, Key = Key₁ | Key₂ | Key₃, with Key₁ consisting of the last 128 bits, Key₂ consisting of the 128 bits before them, and Key₃ consisting of the remaining first 128 or 256 bits.

If not empty, the tweak is partitioned into a sequence of blocks T = T₁ | T₂ | … | Tᵣ, where each of the blocks T₁, T₂, …, Tᵣ₋₁ is of length exactly 16 bytes, and Tᵣ is of length between 1 and 16 bytes.

The plaintext P is partitioned into a sequence of blocks P = P₁ | P₂ | … | Pᵢ, where each of the blocks P₁, P₂, …, Pᵢ₋₁ is of length exactly 128 bytes, and Pᵢ is of length between 1 and 128 bytes.

The ciphertext shall then be computed by the sequence of steps in Table 1 or equivalent. An illustration of these steps (for plaintext of 130 full blocks and one partial block) is provided in Figure 1.

Figure 1, An illustration of EME2-AES
Table 1, The EME2-AES Encryption Procedure

// Process the associated data T to get the 128-bit block T_star
1. if len(T)==0 then T_star = AES-Enc(Key, Key);
2. else
3. Keyi = Mult-by-alpha(Key_i)
4. for i=1 to r-1
5. TT_i = AES-Enc(Key_i, Key;T_i) ⊕ Key;
6. Key_i = Mult-by-alpha(Key_i)
7. if len(T_i)<128 then
8. T_i = T_i | 10...0 // pad T_i to 128 bits
9. Key_i = Mult-by-alpha(Key_i)
10. TT_i = AES-Enc(Key_i, Key;T_i) ⊕ Key;
11. T_star = TT_1 ⊕ TT_2 ⊕ … ⊕ TT_r
// First ECB pass
12. L = Key
13. for i=1 to m-1
14. PPP_i = AES-Enc(Key, L⊕P_i)
15. L = Mult-by-alpha(L)
16. if len(P_m)<128 then PPP_m = P_m | 10…0 // pad P_m to 128 bits
17. elsePPP_m = AES-Enc(Key, L⊕P_m)
// Intermediate mixing
18. MP = PPP_1 ⊕ PPP_2 ⊕ … ⊕ PPP_r ⊕ T_star
19. if len(P_m)<128 then
20. MM = AES-Enc(Key, MP)
21. MC = MC = AES-Enc(Key, MM)
22. else MC = MC = AES-Enc(Key, MP)
23. M = M = MP ⊕ MC
24. for i=2 to m-1
25. if (i-1 mod 128 > 0) then
26. M = Mult-by-alpha(M)
27. CCC_i = PPP_i ⊕ M
28. else
29. MP = PPP_i ⊕ M
30. MC = AES-Enc(Key, MP)
31. M = MP ⊕ MC
32. CCC_i = MC ⊕ M_i
33. if len(P_m)<128 then
34. C_m = P_m ⊕ (MM truncated to len(P_m) bits)
35. CCC_m = C_m | 10…0 // pad C_m to 128 bits
36. else if (m-1 mod 128 > 0) then
37. M = Mult-by-alpha(M)
38. CCC_m = PPP_m ⊕ M
39. else CCC_m = AES-Enc(Key, M⊕PPP_m) ⊕ M
40. CCC_m = MC ⊕ CCC_m ⊕ … ⊕ CCC_m ⊕ T_star
// Second ECB Pass
41. L = Key
42. for i=1 to m-1
43. C_i = AES-Enc(Key,CCC_i) ⊕ L
44. L = Mult-by-alpha(L)
45. if len(P_m)==128 then C_m = AES-Enc(Key,CCC_m) ⊕ L
5.2.3 EME2-AES Decryption

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

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

where \( \text{Key} \) is the 48 or 64 byte EME2-AES key, \( T \) is the value of the associated data, of arbitrary byte length (zero or more bytes), \( C \) is the ciphertext, of length 16 bytes or more, and \( P \) is the plaintext resulting from the operation, of the same byte-length as \( C \).

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

- The key is partitioned into three fields, \( \text{Key} = \text{Key}_1 \| \text{Key}_2 \| \text{Key}_3 \) with \( \text{Key}_2 \) consisting of the last 16 bytes, \( \text{Key}_3 \) consisting of the 16 bytes before them, and \( \text{Key}_1 \) consisting of the remaining first 16 or 32 bytes.
- If not empty, the associated data is partitioned into a sequence of blocks \( T = T_1 \| T_2 \| \ldots \| T_r \), where each of the blocks \( T_1, T_2, \ldots, T_r \) is of length exactly 16 bytes, and \( T_1 \) is of length between 1 and 16 bytes.
- The ciphertext \( P \) is partitioned into a sequence of blocks \( C = C_1 \| C_2 \| \ldots \| C_m \), where each of the blocks \( C_1, C_2, \ldots, C_m \) is of length exactly 16 bytes, and \( C_m \) is of length between 1 and 16 bytes.

The plaintext shall then be computed by the sequence of steps in Table 2, 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.

### Table 2: The EME2-AES Decryption routine

```plaintext
// Process the associated data T to get the 128-bit block T_star
1. if len(T)==0 then T_star = AES-Enc(Key_1, Key_2)
2. else
3. Key_3 = Mult-by-alpha(Key_1)
4. for i=1 to r-1
5. TT_i = AES-Enc(Key_3, Key_2 \& T_i) \& Key_1
6. Key_1 = Mult-by-alpha(Key_1)
7. if len(T_i)<128 then
8. T_i = T_i \& 1010... // pad T_i to 128 bits
9. Key_1 = Mult-by-alpha(Key_1)
10. TT_i = AES-Enc(Key_3, Key_2 \& T_i) \& Key_1
11. T_star = TT_1 \& TT_2 \& \ldots \& TT_r
// First ECB pass
12. L = Key_1
13. for i=1 to m-1
14. \text{C}_i = AES-Dec(Key_1, L\&C_{i-1})
```

Copyright ©<year> IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
15. \( L = \text{Mult-by-alpha}(L) \)
16. if \( \text{len}(C_m) < 128 \) then \( C_{C_m} = C_m | 0...0 \) // pad \( C_m \) to 128 bits
17. else \( C_{C_m} = \text{AES-Dec}(\text{Key}_1, L @ C_m) \)

// Intermediate mixing
18. \( M = C_{C_1} @ C_{C_2} @ ... @ C_{C_m} @ T_{\text{star}} \)
19. if \( \text{len}(C_m) < 128 \) then
20. \( M = \text{AES-Dec}(\text{Key}_1, MC) \)
21. \( MP = MP_1 = \text{AES-Dec}(\text{Key}_1, MM) \)
22. else \( MP = MP_1 = \text{AES-Dec}(\text{Key}_1, MC) \)
23. \( MC = M = MP @ MC \)
24. for \( i = 2 \) to \( m-1 \) then
25. \( MP = \text{AES-Dec}(\text{Key}_1, MC) \)
26. \( M = \text{Mult-by-alpha}(M) \)
27. \( P_{\text{PP}_1} = C_{C_1} @ M \)
28. else
29. \( MC = C_{C_1} @ M_1 \)
30. \( MP = \text{AES-Dec}(\text{Key}_1, MC) \)
31. \( M = MP @ MC \)
32. \( P_{\text{PP}_1} = MP @ M_1 \)
33. if \( \text{len}(C_m) < 128 \) then
34. \( P_{m} = C_m @ (MM \text{ truncated to } \text{len}(C_m) \text{ bits}) \)
35. \( P_{i} = P_{\text{PP}_i} = C_{C_i} @ M_{i} \)
36. else if \( (m-1 \mod 128 > 0) \) then
37. \( M = \text{Mult-by-alpha}(M) \)
38. \( P_{\text{PP}_i} = C_{C_i} @ M_{i} \)
39. else \( P_{\text{PP}_m} = \text{AES-Dec}(\text{Key}_1, M_1 @ C_{C_m}) @ M_1 \)
40. \( P_{\text{PP}_i} = MP @ P_{\text{PP}_i} @ ... @ P_{\text{PP}_m} @ T_{\text{star}} \)

// Second ECB Pass
41. \( L = \text{Key}_2 \)
42. for \( i = 1 \) to \( m-1 \) \n43. \( P_i = \text{AES- Dec}(\text{Key}_1, P_{\text{PP}_i}) @ L \)
44. \( L = \text{Mult-by-alpha}(L) \)
45. if \( \text{len}(C_m) = 128 \) then \( P_m = \text{AES- Dec}(\text{Key}_1, P_{\text{PP}_m}) @ L \)
5.3 The XCB-AES Transform

5.3.1 Definition

The XCB-AES encryption and decryption operations use the AES block cipher encryption functions `AES-enc` and `AES-dec`, as well as the hash function `h` and the pseudorandom function `c`. The variables `H`, `K_e`, `K_d`, and `K_c` are derived from `K`, essentially by running the AES-enc encryption function in counter mode.

Optionally, these values can be stored between evaluations of these algorithms, in order to trade off some storage for a decreased computational load.

Let `k` be the size of the key fed to the AES function (either 128 or 256 bits). The function `c : \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l`, where the output length `l` is bounded by `0 <= l <= 2^{39}`, generates an arbitrary-length output by running the AES-enc function in counter mode, using its 128-bit input as the initial counter value. Its definition is

\[
e(K, W, l) = AES-enc(K, W) | AES-enc(K, incr(W) | . . . msb_{128}(AES-enc(K, incr^{n-1}(W))),
\]

where we make the output length `l` an explicit parameter for clarity; `n = \lceil l/128 \rceil` is the number of 128-bit blocks in the output and `t = l \mod 128` is number of bits in the trailing block. Here the function `incr : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}` is the increment operation that is used to generate successive counter values. This function treats the rightmost 32 bits of its argument as a nonnegative integer with the least significant bit on the right, increments this value modulo 2^{32}. More formally,

\[
incr(X) = X[0; 95] | (X[96; 127] + 1 \mod 2^{32}),
\]

where we rely on the implicit conversion of bit strings to integers.

The functions `h_1` and `h_2` are defined in terms of the underlying hash function `h` as

\[
h_1 (H, Z, B) = h(H, 0^{128} | Z, B | padlen_1(#B))
\]

\[
h_2 (H, Z, B) = h(H, Z | 0^{128} | padlen_2(#B) | len(Z) | 0^{128} | len(B) | padlen_1(#B))
\]

The function `padlen_2(x)` returns the smallest number that can be added to `x` so that the result is a multiple of 128. The function `padlen_1(x)` returns `padlen_2(x) + 128`. These functions can be expressed mathematically as

\[
\text{padlen}_2(x) = 128 \times \text{ceil}(x/128) - x
\]

\[
\text{padlen}_1(x) = 128 \times (1 + \text{ceil}(x/128)) - x = 128 + \text{padlen}_2(x)
\]

where `ceil(x)` is the smallest integer that is larger than `x`. 

Copyright © <year> IEEE. All rights reserved. This is an unapproved IEEE Standards Draft, subject to change.
The function $h : \{0,1\}^{128} \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{128}$ is defined by $h(H, A, C) = X_{m+n+1}$, where $a$ and $c$ are multiples of 128 within the interval $[128,2^{39})$ and the variables $X_i \in \{0, 1\}^{128}$ for $i = 0, \ldots, m+n+1$ are defined as

$$X_i = 0 \quad \text{for } i = 0$$

$$X_i = (X_{i-1} \oplus A_{i-1}) \cdot H \quad \text{for } i = 1, \ldots, m - 1$$

$$X_i = (X_{m+i} \oplus A_m) \cdot H \quad \text{for } i = m$$

$$X_i = (X_{m+n+i} \oplus C_m) \cdot H \quad \text{for } i = m + 1, \ldots, m + n - 1$$

$$X_i = (X_{m+n+i} \oplus C_n) \cdot H \quad \text{for } i = m + n$$

$$X_i = (X_{m+n+i} \oplus (\text{len}(A) \oplus \text{len}(C))) \cdot H \quad \text{for } i = m + n + 1.$$

Here we let $A_i$ denote the 128-bit substring $A[128*(i-1); 128*i-1]$, and let $C_i$ denote $C[128*(i-1); 128*i-1]$. In other words, $A_i$ and $C_i$ are the $i$th blocks of $A$ and $C$, respectively, if those bit strings are decomposed into 128-bit blocks. This function is identical to the universal hash function that is used as a component of the Galois/Counter Mode (GCM) of Operation [SP800-38D]. (It is equivalent to the function used in Step 5 of Algorithm 4 of that specification, but please note that it is different than GHASH as defined in that document.)
5.3.2 Multiplication in GF($2^{128}$)

The multiplication operation of two 16-byte values $X, Y \in \text{GF}(2^{128})$ is mathematically equivalent to an operation on bit vectors. The result $Z = X \cdot Y$ is also an element of $\text{GF}(2^{128})$. The input values are first converted into a byte array $X[i], i = 0, 1, \ldots, 15$, where the leftmost bit is $X[0]$, and the rightmost bit is $X[127]$.

The multiplication operation is defined by the following, or a mathematically equivalent, procedure:

```c
int multiply(byte array X[], byte array Y[], byte array Z[])
{
    // initialize z to the all-zero element */
    z[0] = 0
    endfor
    for i=0 to 15    /* loop over bytes of y */
        mask = 128;
        while mask > 0       /* loop over bits in byte */
            /* if masked bit is set, add in terms from x */
            set b to y[i] & mask
            if b != 0
                for j=0 to 15
                    set z[j] to z[j] ^ x[j];
                endfor
                mask = mask / 2;
            endif
        endwhile
        /* now execute LFSR shift on x */
        set msb to x[15] & 1
        for j=15 down to 1
            set b to x[j-1] & 1
            set x[j] to (x[j] / 2) + b * 128
        endfor
        set x[0] to x[0] / 2
        if msb = 1
            set x[0] to x[0] ^ R
        endif
    endfor
    return z
}
```

NOTE - The multiplication operation uses the special element $R = \text{11100001|0}$\text{120}. The function rightshift() moves the bits of its argument one bit to the right. More formally, whenever $W = \text{rightshift}(V)$, then $W[i] = V[i-1]$ for $1 \leq i \leq 127$ and $W[0] = 0$. 


5.3.3 The XCB-AES Encryption Operation

The XCB-AES encryption operation for an m-bit block P is modeled with this equation:

\[ \text{CT} \leftarrow \text{XCB-AES-enc}(K,P,Z) \]

where:

- \( K \) is either the 128 or 256 bit XCB-AES key
- \( Z \) is the value of the associated data, of arbitrary byte length (zero or more bytes)
- \( P \) is a block of plaintext of \( m \) bits where \( m \in [128,2^{32}] \)
- \( \text{CT} \) is the block of 128 bits of ciphertext resulting from the operation

The ciphertext shall then be computed by the following or an equivalent sequence of steps (see Figure 1):

- \( H \leftarrow \text{AES-enc}(K,0^{128}) \)
- \( K_e \leftarrow \msb(AES-enc(K,0^{125}|001) \oplus AES-enc(K,0^{125}|010)) \)
- \( K_d \leftarrow \msb(AES-enc(K,0^{125}|011) \oplus AES-enc(K,0^{125}|100)) \)
- \( K_c \leftarrow \msb(AES-enc(K,0^{125}|101) \oplus AES-enc(K,0^{125}|110)) \)
- \( A \leftarrow P[m-128; m-1] \)
- \( B \leftarrow P[0; m-127] \)
- \( C \leftarrow \text{AES-enc}(K_c,A) \)
- \( D \leftarrow C \oplus h_1(H,Z,B) \)
- \( E \leftarrow B \oplus c(K_c,D,\#B) \)
- \( F \leftarrow D \oplus h_2(H,Z,E) \)
G ← AES-dec(K_d,F)
CT ← E(G)
Figure 2, An illustration of XCB-AES

5.3.4 The XCB-AES Decryption Operation

The XCB-AES decryption operation for an m-bit block $P$ is modeled with this equation:
P ← XCB-AES-dec(K,CT,Z)

where:

K is either the 128 or 256 bit XCB-AES key

Z is the value of the associated data, of arbitrary byte length (zero or more bytes)

CT is a block of ciphertext of m bits where m ∈ [128,2^{32}]

P is the block of 128 bits of plaintext resulting from the operation

The plaintext shall then be computed by the following or an equivalent sequence of steps:

$H ← \text{AES-enc}(K,0^{128})$

$K_e ← \text{msb}_k(\text{AES-enc}(K,0^{128}|001) || \text{AES-enc}(K,0^{128}|010))$

$K_d ← \text{msb}_k(\text{AES-enc}(K,0^{128}|011) || \text{AES-enc}(K,0^{128}|100))$

$K_c ← \text{msb}_k(\text{AES-enc}(K,0^{128}|101) || \text{AES-enc}(K,0^{128}|110))$

$G ← P[m-128; m-1]$

$E ← P[0; m-127]$

$F ← \text{AES-enc}(K_d,A)$

$D ← F \oplus h_2(H,Z,E)$

$B ← E \oplus c(K_c,D,#B)$

$C ← D \oplus h_1(H,Z,B)$

$A ← \text{AES-dec}(K_c,F)$

$P ← B|A$
6. Use of Wide-Block Encryption for Storage

The encryption and decryption procedures described in 5.2.2 and 5.3.2 use AES as the basic building block with a key of either 384 or 512 bits. For completeness, the first mode shall be referred to as EME2-AES-384 and the second as EME2-AES-512.

The encryption and decryption procedures described in 5.3.2 and 5.3.4 use AES as the basic building block with a key of either 128 or 256 bits. For completeness, the first mode shall be referred to as XCB-AES-128 and the second as XCB-AES-256.

To be compliant with this standard, the implementation shall support at least one of the modes described in the standard.

Exporting and archiving keys used for wide-block encryption of storage can be done using the Key backup Structure defined in clause 7 of IEEE Std 1619-2007. Key scope defines the range of data encrypted with a single key. As defined in clause 7.1.4 of IEEE Std 1619-2007, the Key Scope is represented by three integers: the value of the particular associated data corresponding to the data unit in the sequence of data units encrypted by this key, the size in bits of each data unit, and the number of units to be encrypted/decrypted under the control of this key. An implementation compliant with this standard may or may not support multiple data unit sizes.

In an application of this standard to sector-level encryption of a disk, the data unit typically corresponds to a logical block, the key scope typically includes a range of consecutive logical blocks on the disk, and the associated data value corresponding to the first data unit in the scope typically corresponds to the Logical Block Address (LBA) associated with the logical block in the range. The associated data values are assigned consecutively, starting from an arbitrary non-negative integer. When encrypting an associated data value using AES, the associated data value is first converted into a little-endian byte array. For example the associated data value 0x123456789a corresponds to byte array 0xa9, 0x78, 0x56, 0x34, 0x12.

Key used for wide-block encryption of storage shall not be associated with more than one key scope.

NOTE: The reason of the above restriction is that encrypting more than one block with the same key and the same index introduces security vulnerabilities that might potentially be used in an attack on the system. In particular, key reuse enables trivial cut-and-paste attacks.
Annex A Bibliography

Deleted: 2
Fabio Maino 6/10/08 3:34 PM
Deleted: May
Fabio Maino 6/10/08 3:34 PM
Comment:
Fabio Maino 6/16/08 4:07 PM
Annex B Rationale and Security Concerns
Annex C Implementation in C
Annex D Test Vectors

D.1 EME2-128 Test Case

[TBD]

D.2 XCB-AES-128 Test Case

<table>
<thead>
<tr>
<th>K</th>
<th>000102030405060708090a0b0c0d0e0f</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>c6a13b3787f5b826f4f8f1621e8d879</td>
</tr>
<tr>
<td>1</td>
<td>734613959c041e497bbe365f42d0a</td>
</tr>
</tbody>
</table>

| F   | 734613959c041e497bbe365f42d0a   |
| L   | b9ad2b2a346ac238505d365e9c7fc56 |

| 0   | 00000000000000000000000000000000 |
| A   | 00000000000000000000000000000000 |
| B   | 00000000000000000000000000000000 |
| C   | 00000000000000000000000000000000 |
| D   | 00000000000000000000000000000000 |
| E   | 00000000000000000000000000000000 |
| F   | 00000000000000000000000000000000 |
| G   | 00000000000000000000000000000000 |

| X   | 00000000000000000000000000000000 |
| Y   | 00000000000000000000000000000000 |
| Z   | 00000000000000000000000000000000 |

| A   | 00000000000000000000000000000000 |
| B   | 00000000000000000000000000000000 |
| C   | 00000000000000000000000000000000 |
| D   | 00000000000000000000000000000000 |
| E   | 00000000000000000000000000000000 |
| F   | 00000000000000000000000000000000 |
| G   | 00000000000000000000000000000000 |

| X   | 00000000000000000000000000000000 |
| Y   | 00000000000000000000000000000000 |
| Z   | 00000000000000000000000000000000 |

| A   | 00000000000000000000000000000000 |
| B   | 00000000000000000000000000000000 |
| C   | 00000000000000000000000000000000 |
| D   | 00000000000000000000000000000000 |
| E   | 00000000000000000000000000000000 |
| F   | 00000000000000000000000000000000 |
| G   | 00000000000000000000000000000000 |

| X   | 00000000000000000000000000000000 |
| Y   | 00000000000000000000000000000000 |
| Z   | 00000000000000000000000000000000 |

| A   | 00000000000000000000000000000000 |
| B   | 00000000000000000000000000000000 |
| C   | 00000000000000000000000000000000 |
| D   | 00000000000000000000000000000000 |
| E   | 00000000000000000000000000000000 |
| F   | 00000000000000000000000000000000 |
| G   | 00000000000000000000000000000000 |

| X   | 00000000000000000000000000000000 |
| Y   | 00000000000000000000000000000000 |
| Z   | 00000000000000000000000000000000 |

| A   | 00000000000000000000000000000000 |
| B   | 00000000000000000000000000000000 |
| C   | 00000000000000000000000000000000 |
| D   | 00000000000000000000000000000000 |
| E   | 00000000000000000000000000000000 |
| F   | 00000000000000000000000000000000 |
| G   | 00000000000000000000000000000000 |

| X   | 00000000000000000000000000000000 |
| Y   | 00000000000000000000000000000000 |
| Z   | 00000000000000000000000000000000 |