Document Number: XXXX-00-0-Y-Oct04

Draft Proposal for Tweakable Wide-block Encryption

Draft 1.00:01

October 20, 2004

Sponsor
IEEE P1619

Abstract: We describe the EME-32-AES tweakable block cipher and its use of for encryption of storage. EME-32-AES is a tweakable block cipher that acts on wide blocks of 512 bytes, and it uses as subroutine the AES block cipher (that acts on blocks of 16 bytes). Specifically, EME-32-AES encrypts and decrypts wide blocks of 512 bytes, under the control of a secret AES *key* and a non-secret 16-byte *tweak*. EME-32-AES is a concrete instantiation of the EME mode of operation, as described in reference [HR04]. When used to encrypt storage data, the tweak value is computed as the logical position of the current wide block within the scope of the current key.

The motivating application for EME-32-AES is encryption of storage at the sector level. In this application, sectors are typically of length 512 bytes, and one may use the address of the sector on the disk as the tweak value.

Keywords: Encryption, storage.

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art.

Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

1. Status summary

Contacts

This proposal editor:
Shai Halevi
IBM T.J. Watson Research Center.
19 Skyline Drive
Hawthorne, NY 10532
Tel: +1.914.784.7653
Fax: +1.914.784.6205
Email: shaih@alum.mit.edu

Table of contents

List of figures

List of tables

2. Overview

2.1 Scope and purpose

The purpose of this document is to specify the EME-32-AES transform and its use for encryption of data at rest. The EME-32-AES transform acts on wide blocks of 512 bytes, under the control of a secret key and a non-secret tweak. It is implemented as a mode of operation for the AES block cipher (that has blocks of size 16 bytes). The security goal of EME-32-AES states that it should look like a block cipher (with "wide blocks" of size 512 bytes). Moreover, using the same key with different tweaks should look like using completely independent keys.

EME-32-AES is a concrete instantiation of the EME mode of operation, which is described in reference [HR04]. The EME mode of operation uses a block cipher with n-byte blocks, and turns it into a tweakable cipher with blocks of size upto $8n^2$ bytes. It is proven in [HR04] that the EME mode indeed achieves the stated security goal, assuming that the underlying block cipher is secure.

An example application for this transform is encryption of storage at the sector level, where the encrypting device is not aware of high-level concepts like files and directories. The disk is often partitioned into fixed-length sectors (typically 512 bytes), and the encrypting device is given one sector at a time, in arbitrary order, to encrypt or decrypt. The device needs to operate on sectors as they arrive, independently of the rest. Moreover, the ciphertext must have the same length as its plaintext. On the other hand, it is possible to vary the encryption/decryption process, based on the location on the disk where the ciphertext is stored. The dependency on the location allows that identical plaintext sectors stored at different places on the disk will have unrelated ciphertexts.

This document includes the description of the EME-32-AES transform itself (in both encryption and decryption modes), as well as how it should be used for encryption of data at rest. The scope is limited to encryption of storage data, consisting of an integral number of 512-byte blocks. Throughout this document, a block of 512 consecutive bytes is referred to as a "wide block".

2.2 Related work

Efforts to construct a block cipher with a large block-size from one with a smaller block-size go back to Luby and Rackoff [LR88], whose work can be viewed as doubling the block-size. They were also the first to formally define the security goal of a block cipher. The first attempt to directly construct a cipher with very large blocks from one with small blocks is due to Zheng, Matsumoto, and Imai [ZMI89]. Naor and Reingold describe in [NR98,NR99] an elegant approach doing just that, using a layer of ECB encryption that is "sandwiched" between two layers of non-cryptographic hashing. In practice, however, instantiating the layers of "non-cryptographic hashing" turn out to be problematic. A mode of operation similar to EME (called CMC) was proposed by Halevi and Rogaway in [HR03]. The main difference between CMC and EME is that the latter is parallelizable, whereas the former is not. A different approach for constructing a block cipher with large blocks is to build it cipher from scratch, as with BEAR, LION [AB96], and Mercy [C00].

The formal definition of the security goal of a tweakable block-cipher is due to Liskov, Rivest, and Wagner [LRW02], where they also show how (narrow-block) tweakable ciphers can be built from standard block ciphers. An earlier work by Schroeppel suggested the idea of a tweakable block-cipher, by designing a cipher that natively incorporates a tweak [S98].

3. References

[R1] ANSI/ISO 9899-1990, Programming Language—C.[1,2]

[R2] NIST FIPS-197, Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard.[3]

All the standards listed are normative references. Informative references are given in Annex A. At the time of publication, the editions indicated were valid.

1 Replaces ANSI X3.159-1989.
2 ISO documents are available from ISO Central Secretariat, 1 rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzer-land/Suisse; and from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036-8002, USA
3 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/

4. Definitions

4.1 Conformance levels

4.1.1 expected: A key word used to describe the behavior of the hardware or software in the design models *assumed* by this specification. Other hardware and software design models may also be implemented.

4.1.2 may: A key word indicating flexibility of choice with *no implied preference*.

4.1.3 shall: A key word indicating a mandatory requirement. Designers are *required* to implement all such mandatory requirements.

4.1.4 should: A key word indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase *is recommended*.

4.1.5 reserved fields: A set of bits within a data structure that is defined in this specification as reserved, and is not otherwise used. Implementations of this specification shall zero these fields. Future revisions of this specification, however, may define their usage.

4.1.6 reserved values: A set of values for a field that are defined in this specification as reserved, and are not otherwise used. Implementations of this specification shall not generate these values for the field. Future revisions of this specification, however, may define their usage.

NOTE —These conformance definitions are used throughout IEEE standards and should therefore never be changed.

4.2 Glossary of terms

4.2.1 byte: Eight bits of data, used as a synonym for octet.

4.2.2 doublet: Two bytes of data.

4.2.3 quadlet: Four bytes of data.

4.2.4 octlet: Eight bytes of data.

3.2.5 block: Sixteen bytes of data.

3.2.6 wide block: Five hundred and twelve bytes of data.

4.3 Acronyms and abbreviations

IEEE The Institute of Electrical and Electronics Engineers, Inc.

4.4 Numerical values

Decimal, hexadecimal, and binary numbers are used within this document. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their usual 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9,A-F) digits followed by the subscript 16, except in C-code contexts, where they are written as `0x123EF2` etc. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number "26" may also be represented as "$1A_{16}$" or "11010_2".

4.5 Field names

This document describes values that are in memory-resident or control-and-status registers (CSRs). For clarity, names of these values have an italics font and contain the context as well as field names, as illustrated in Table 4.1.

Table 4.1— Names of registers and fields

Name	Description
MoverCsr.control	The mover's control register.
Command.code.	The code field within a command entry
Status.count	The count field within a status entry

Note that run-together names like *"MoverCsr*" are preferred because they are more compact than under-score-separated names (like *"Mover_Csr*"). The use of multiword names with spaces (like "Mover CSR" is avoided, to avoid confusion between commonly used capitalized key words and the capitalized word used at the start of each sentence. Capitalization may, however, be useful for differentiating between different types of key words. For example: the upper case *MoverCs*r, *Comman*d, and *Status* names refer to CSR registers and the lower case *contro*l, *cod*e, and *count* names refer to fields within these registers.

4.6 Notations for block encryption and decryption

Subject to the procedures described in publication [R2] (AES), we denote by C = AES-enc(K; P) the operation of applying the encryption procedure from [R2], when K is an array of bytes that is used as the encryption key, and P is a block of 16 bytes. The result of this operation is a block of 16 bytes, which is denoted C. Similarly, we denote by P = AES-dec(K; C) the operation of applying the decryption procedure from [R2], when K is an array of bytes that is used as the decryption key, and C is a block of 16 bytes. The result of this operation is a block of 16 bytes, which is denoted P.

4.7 C-code notation

The behavior of many commands is frequently specified by C code, such as in Equation 4.1. To differentiate this code from textual descriptions, such C code listings are formatted using a fixed-width Courier font. Similar C-code segments are included within some figures.

```
// Return maximum of a and b values
Max(a,b) {
  if (a<b)
    return(LT);
  if (a>b)
    return(GT);
  return(EQ);
}
```

4.1

Since the meaning of many C code operators are not obvious to the casual reader, their meanings are summarized in Table 4.2.

Table 4.2—C code expressions

Expression	Description
`~i`	Bitwise complement of integer *i*
`i^j`	Bitwise EXOR of integers *i* and *j*
`i&j`	Bitwise AND of integers *i* and *j*
`i<<j`	Left shift of bits in *i* by value of *j*
`i*j`	Arithmetic bmultiplication of integers *i* and *j*
`!i`	Logical negation of Boolean value *i*
`i&&j`	Logical AND of Boolean *i* and *j* values
`i||j`	Logical OR of Boolean *i* and *j* values
`i^= j`	Equivalent to i= i^j.
`i==j`	Equality test, true if *i* equals *j*
`i!=j`	Equality test, true if *i* does not equal *j*
`i<j`	Inequality test, true if *i* is less than *j*
`i>j`	Inequality test, true if *i* is greater than *j*

5. The EME-32-AES transform

In this section we describe the EME-32-AES transform itself. That is, the procedures to be followed when encrypting or decrypting a wide block, given the secret key and the public tweak value. A pictorial illustration of the encryption procedure is provided in Figure 5.1. In that figure, all the lines represent 16-byte blocks, and the boxes represent the AES encryption procedure, all using the same key. The symbol `⊕' denotes bit-wise exclusive-or (xor) of two 16-byte blocks, and multiplications by powers of two are implemented via the procedure described in Section 5.1. A complete description can be found in Section 5.2.

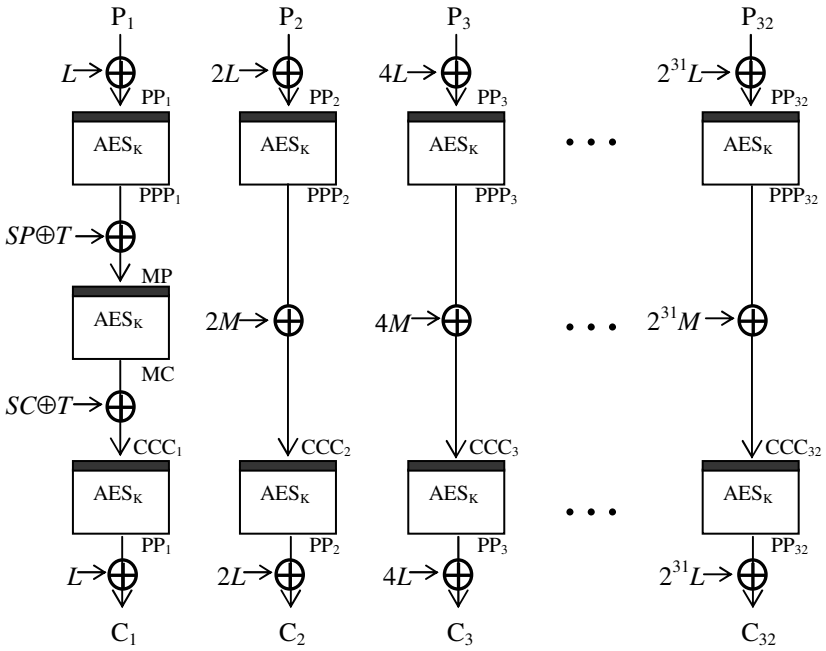

Figure 5.1. An illustration of EME encryption, $C_{1..32} = EME_K(T; P_{1..32})$. We set $L=2\cdot AES_K(0)$, $SP=PPP_2 \oplus \ldots \oplus PPP_{32}$, $M=MP \oplus MC$, and $SC=CCC_2 \oplus \ldots \oplus CCC_{32}$

5.1 Multiplication by two in the finite field GF(2^{128})

We now describe a procedure for multiplying a 16-byte block by the constant "two" in the finite field GF(2^{128}). Both the input and the output of this procedure are 16-byte blocks. When these blocks are interpreted as binary polynomials of degree 127, the procedure computes $output = x \cdot input$ modulo P_{128}, where P_{128} is the polynomial $P_{128}(x) = x^{128} + x^7 + x^2 + x + 1$. The procedure is implemented as follows:

First, we compute a *Mask* block that depends on the highest bit in the last byte of the input block. If that bit is zero (i.e., if the value of the last byte is less than 128), then the mask is set to the all-zero block. If the bit is one (i.e., if the value of the last byte is more than 127), then the value of the *Mask* block is set to the constant

$$Mask = 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 87_{16}$$

Namely, in the latter case the first byte of the *Mask* block is set to the constant 135 (hexadecimal 87_{16}), and all the other bytes are set to zero.

Then, the input block is copied to a temporary block *Temp*, where the entire block is shifted by one bit toward the highest bit. Specifically, let the value of the i^{th} input byte be b_i and the value of the $(i\text{-}1)^{st}$ input byte be b_{i-1}. If b_{i-1} is less than 128, then the i^{th} byte in the block *Temp* is set to $2b_i$ modulo 256, and if b_{i-1} is

more than 127 then the i^{th} byte in the block *Temp* is set to $2b_i+1$ modulo 256. Finally, the output block is set to the bit-wise exclusive-or (xor) of the blocks *Temp* and *Mask*. A C-code for this procedure is given in Figure 5.2.

```
void multByTwo(unsigned char output[16], unsigned char input[16])
{
   int j;
   unsigned char temp[16];   /* a temporary array, just in case input and */
                             /* output point to the same memory location  */
   temp[0] = 2 * input[0];
   if (input[15] >= 128) temp[0] ^= 135;
   for (j=1; j<16; j++) {
      temp[j] = 2 * input[j];
      if (input[j-1] >= 128) temp[j] += 1;
   }
   for (j=0; j<16; j++) output[j] = temp[j];
}
```

Figure 5.2. A "C" code for the `multByTwo` procedure.

5.2 The EME encryption procedure

The EME encryption procedure takes as input an AES key K, a 512-byte wide block $\boldsymbol{P}$ (for Plaintext) and a 16-byte block T (for Tweak). It produces as output a 512-byte wide block $\boldsymbol{C}$ (for Ciphertext). The procedure works as follows:

1. A 16-byte block L is computes as $L = 2 \cdot \text{AES-enc}(K; \underline{0})$, where AES-enc is the AES encryption procedure using K as the key, $\underline{0}$ is a 16-byte block with all the bytes set to zero, and the multiplication by 2 is done using the procedure from Section 5.1.

2. The input wide block $\boldsymbol{P}$ is broken into 32 blocks $P_1\ P_2\ \ldots\ P_{32}$, where P_1 consists of the first 16 bytes in $\boldsymbol{P}$, P_2 consists of the next 16 bytes, etc. In general, for j=1, 2, … 32, the block P_j consists of bytes number 16j−15 through 16j (indexing starts at one).

3. For j=1, 2, … 32, a 16-byte block PP_j is computed as $PP_j = P_j \oplus (2^{j-1} \cdot L)$, where `⊕' denotes bit-wise exclusive or, and multiplication by powers of two is done via repeated applications of the procedure from Section 5.1. For example, the first three blocks are computed as $PP_1 = P_1 \oplus L$, $PP_2 = P_2 \oplus (2 \cdot L)$, and $PP_3 = P_3 \oplus (2 \cdot 2 \cdot L)$.

4. For j=1, 2, … 32, a 16-byte block PPP_j is computed as $PPP_j = \text{AES-enc}(K; PP_j)$.

5. A 16-byte block SP is computed as the xor-sum of the blocks PPP_2 through PPP_{32}, $SP = PPP_2 \oplus PPP_3 \oplus \ldots \oplus PPP_{32}$. Then a 16-byte block MP is computed as $MP = PPP_1 \oplus SP \oplus T$ (T is the "tweak").

6. A 16-byte block MC is computed as $MC = \text{AES-enc}(K; MP)$. Then a 16-byte block M is computed as the bit-wise exclusive-or (xor) of MP and MC, $M = MP \oplus MC$.

7. For j=2, 3, … 32, a 16-byte block CCC_j is computed as $CCC_j = PPP_j \oplus (2^{j-1} \cdot M)$, where multiplication by powers of two is done via repeated applications of the procedure from Section 5.1. For example, the first two blocks are computed as $CCC_2 = PPP_2 \oplus (2 \cdot M)$, and $CCC_3 = PPP_3 \oplus (2 \cdot 2 \cdot M)$.

8. A 16-byte block SC is computed as the xor-sum of the blocks CCC_2 through CCC_{32}, namely $SC = CCC_2 \oplus CCC_3 \oplus \ldots \oplus CCC_{32}$. Then, a 16-byte block CCC_1 is computed as $CCC_1 = MC \oplus SC \oplus T$, where T is the input "tweak".

9. For j=1, 2, … 32, a 16-byte block CC_j is computed as CC_j = AES-enc(K; CCC_j), and then a 16-byte block C_j is computed as $C_j = CC_j \oplus (2^{j-1} \cdot L)$, where multiplication by powers of two is done via repeated applications of the procedure from Section 5.1.

10. Finally, the 32 blocks C_1 through C_{32} are concatenated to form the output wide block $\boldsymbol{C}$. That is, the first 16 bytes in $\boldsymbol{C}$ come from C_1, the next 16 bytes from C_2, etc. In general, for j=1, 2, … 32, bytes number 16j−15 through 16j in $\boldsymbol{C}$ come from block C_j (indexing starts at one).

A C-code for this procedure is given in Figure 5.3. This procedure employs many calls to the AES encryption procedure, all using the same key K. Since the key-setup time for AES is significant, an implementation of EME **may** perform it just once, and then use the resulting key-schedule in all the encryption calls. Moreover, as the computation of the block L does not depend on the input wide block $\boldsymbol{P}$ or the input tweak T, an implementation **may** compute the value of L off-line, before learning either $\boldsymbol{P}$ or T. In fact, an implementation **may** even store the key-schedule and the block L between different EME calls.

5.3 The EME decryption procedure

The EME decryption procedure takes as input an AES key K, a 512-byte wide block $\boldsymbol{C}$ (for Ciphertext) and a 16-byte block T (for Tweak). It produces as output a 512-byte wide block $\boldsymbol{P}$ (for Plaintext). The procedure is very similar to the encryption procedure. In fact, a "C" code for the decryption procedure can be obtained from the "C" code in Figure 5.3, simply by replacing all but the first call to `encryptAES` (i.e. all calls except the call to `encryptAES(zero, K, zero)`) by calls to `decryptAES`. The decryption procedure works as follows:

1. A 16-byte block L is computes as $L = 2 \cdot$ AES-enc(K; $\underline{0}$), where AES-enc is the AES encryption procedure using K as the key, $\underline{0}$ is a 16-byte block with all the bytes set to zero, and the multiplication by 2 is done using the procedure from Section 5.1.

2. The input wide block $\boldsymbol{C}$ is broken into 32 blocks $C_1\ C_2\ \ldots\ C_{32}$, where C_1 consists of the first 16 bytes in $\boldsymbol{C}$, C_2 consists of the next 16 bytes, etc. In general, for j=1, 2, … 32, the block C_j consists of bytes number 16j−15 through 16j (indexing starts at one).

3. For j=1, 2, … 32, a 16-byte block CC_j is computed as $CC_j = C_j \oplus (2^{j-1} \cdot L)$, where multiplication by powers of two is done via repeated applications of the procedure from Section 5.1.

4. For j=1, 2, … 32, a 16-byte block CCC_j is computed as CCC_j = AES-dec(K; CC_j).

5. A 16-byte block SC is computed as the xor-sum of the blocks CCC_2 through CCC_{32}, namely $SC = CCC_2 \oplus CCC_3 \oplus \ldots \oplus CCC_{32}$.

6. A 16-byte block MC is computed as $MP = CCC_1 \oplus SC \oplus T$, where T is the input "tweak". Then a 16-byte block MP is computed as MP = AES-dec(K; MC). Next a 16-byte block M is computed as the bit-wise exclusive-or (xor) of MC and MP, $M = MC \oplus MP$.

7. For j=2, 3, … 32, a 16-byte block PPP_j is computed as $PPP_j = CCC_j \oplus (2^{j-1} \cdot M)$, where multiplication by powers of two is done via repeated applications of the procedure from Section 5.1.

8. A 16-byte block *SP* is computed as the xor-sum of the blocks PPP_2 through PPP_{32}, namely $SP = PPP_2 \oplus PPP_3 \oplus \ldots \oplus PPP_{32}$. Then, a 16-byte block PPP_1 is computed as $PPP_1 = MP \oplus SP \oplus T$, where *T* is the input "tweak".

9. For j=1, 2, … 32, a 16-byte block PP_j is computed as PP_j = AES-dec(K; PPP_j), and then a 16-byte block P_j is computed as $P_j = PP_j \oplus (2^{j-1} \cdot L)$, where multiplication by powers of two is done via repeated applications of the procedure from Section 5.1.

10. Finally, the 32 blocks P_1 through P_{32} are concatenated to form the output wide block ***P***. That is, the first 16 bytes in ***P*** come from P_1, the next 16 bytes from P_2, etc. In general, for j=1, 2, … 32, bytes number 16j−15 through 16j in ***P*** come from block P_j (indexing starts at one).


```
/* The function encryptAES implements the AES encryption procedure. It is
 * assumed that this procedure has external means for determining the key
 * length.
 */
extern void encryptAES(unsigned char out[16],
                       unsigned char key[], unsigned char in[16]);

/* The function xorBlocks computes bit-wise exclusive-or of two blocks */
extern void xorBlocks(unsigned char out[16],
                      unsigned char in1[16], unsigned char in2[16]);

/* The function multByTwo is described in Figure 5.2 */
extern void multByTwo(unsigned char out [16], unsigned char in[16]);

void encryptEME(unsigned char C[512], unsigned char K[],
                unsigned char T[16], unsigned char P[512])
{
  int i,j;
  unsigned char L[16], M[16], MP[16], MC[16];
  unsigned char temp[512];    /* a temporary array, just in case P and C */
                              /*       point to the same memory location */
  unsigned char zero[16] = {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};

  encryptAES(zero, K, zero);                  /* set L = 2*AES-enc(K; 0) */
  multByTwo(L, zero);

  for (j=0; j<32; j++) {
    xorBlocks(&temp[j*16],&P[j*16], L);       /* PPj = 2**(j-1)*L xor Pj */
    encryptAES(&temp[j*16], K, &temp[j*16]);  /* PPPj = AES-enc(K; PPj)  */
    multByTwo(L, L);
  }

  xorBlocks(MP, temp, T);                     /* MP =(xorSum PPPj) xor T */
  for (j=1; j<32; j++)
    xorBlocks(MP, MP, &temp[j*16]);
  encryptAES(MC, K, MP);                      /* MC = AES-enc(K; MP)     */
  xorBlocks(M, MP, MC);                       /* M = MP xor MC           */

  for (j=1; j<32; j++) {
    multByTwo(M, M);
    xorBlocks(&temp[j*16],&temp[j*16],M);  /* CCCj = 2**(j-1)*M xor PPPj */
  }
  xorBlocks(temp, MC, T);           /* CCC1 = (xorSum CCCj) xor T xor MC */
  for (j=1; j<32; j++)
    xorBlocks(temp, temp, &temp[j*16]);

  multByTwo(L, zero);                       /* reset L = 2*AES-enc(K; 0) */
  for (j=0; j<32; j++) {
    encryptAES(&temp[j*16], K, &temp[j*16]);  /* CCj = AES-enc(K; CCCj)  */
    xorBlocks(&C[j*16], &temp[j*16],  L);     /* Cj = 2**(j-1)*L xor CCj */
    multByTwo(L, L);
  }
}
```

Figure 5.3. A “C” code for the EME encryption procedure.

6. Using EME-32-AES for encryption of storage

The scope of this document is limited to direct application of the EME-32-AES transform to encrypt or decrypt data at rest, when this data consists of an integral number of wide blocks, each of size 512 bytes. To use this standard, an AES key **shall** be associated with an ordered sequence of wide blocks, numbered consecutively 1 through N, where 1 is the index of the first logical wide block for this key, and N is the index of the last one. The sequence of wide blocks that are associated with an AES key is related to the *SCOPE* of that key, as defined in [BACKUP]. In order to encrypt or decrypt a wide block using an AES key, the index of the wide block within the scope of the key must be known.

Key Scope is defined in [BACKUP] as being represented by two integers which represent the "LBA" or Logical Byte Address of the start of the storage to be encrypted in bytes, and the number of bytes to be encrypted. To be valid for use with EME-32-AES, the number of bytes to be encrypted in the key scope **shall** be a multiple of 512. Also, since the index N of the last block in the range must be less than 2^{128}, the number of bytes in the scope **shall** be less than or equal to $512(2^{128}-1)$. Each wide block (of 512 bytes) within the key scope is then associated with a logical index *J*, which may be viewed as the number of wide blocks from the beginning of the key scope, with the first wide block being numbered 1.

In a typical application for storage encryption, the scope of a key typically includes a range of logically consecutive sectors on the disk. Start location of the key scope would be the location of the beginning of the first sector in the range. The alignment of key scopes with integral numbers of consecutive storage sectors is **recommended**, but **not mandated**.

To encrypt a plaintext wide block with index J, the positive integer J is first encoded as a 16-byte block T_J, as explained in Section 6.1 below. Then the EME-32-AES encryption transform is applied to this wide block, using the given key and the tweak value T_J, as described in Section 5.2. The result of the transformation is a ciphertext wide block. Similarly, to decrypt a ciphertext wide block with index J, the positive integer J is encoded as a 16-byte block T_J, then the EME-32-AES decryption transform is applied to this wide block using the given key and the tweak value T_J, as described in Section 5.3, and the result is a plaintext wide block.

It is stressed that a single AES secret key **shall not** be associated with more than one scope. The reason is that encrypting more than one wide block with the same AES key and the same index introduces security vulnerabilities that can potentially be used in an attack on the system.

6.1 Encoding the tweak values

A positive integer J (smaller than 2^{128}) is encoded as a 16-byte block T using big-endian notation. That is, the integer J is represented in base-256 notation, where the most significant byte is stored as the first byte in the block T and the least significant byte is stored as the last byte. Using "C" notations, we view T as an array of sixteen `unsigned char`, indexed from 0 (first) to 15 (last), with each byte representing a number between 0 and 255, then the integer J is

$$J = T[0]\cdot 256^{15} + T[1]\cdot 256^{14} + T[2]\cdot 256^{13} + T[3]\cdot 256^{12} + T[4]\cdot 256^{11} + T[5]\cdot 256^{10} + T[6]\cdot 256^{9} + T[7]\cdot 256^{8} + T[8]\cdot 256^{7} + T[9]\cdot 256^{6} + T[10]\cdot 256^{5} + T[11]\cdot 256^{4} + T[12]\cdot 256^{3} + T[13]\cdot 256^{2} + T[14]\cdot 256 + T[15]$$

Annexes

Annex A: Bibliography (informative)

[AB96] R. Anderson and E. Biham. "Two practical and provably secure block ciphers: BEAR and LION." In *Fast Software Encryption, Third International Workshop*, volume 1039 of Lecture Notes in Computer Science, pages 113-120. Springer-Verlag, 1996.

[C00] P. Crowley. "Mercy: A fast large block cipher for disk sector encryption." In *Fast Software Encryption: 7th International Workshop*, volume 1978 of Lecture Notes in Computer Science, pages 49-63, Springer-Verlag, 2000.

[HR03] S. Halevi and P. Rogaway. "A tweakable enciphering mode." In *Advances in Cryptology – CRYPTO '03*, volume 2729 of Lecture Notes in Computer Science, pages 482-499. Springer-Verlag, 2003.

[HR04] S. Halevi and P. Rogaway. "A parallelizable enciphering mode." The RSA conference - Cryptographer's track, RSA-CT '04. LNCS vol. 2964, pages 292-304. Springer-Verlag, 2004.

[LRW02] M. Liskov, R. Rivest, and D. Wagner. "Tweakable block ciphers." In *Advances in Cryptology – CRYPTO '02*, volume 2442 of Lecture Notes in Computer Science, pages 31-46. Springer-Verlag, 2002.

[LR88] M. Luby and C. Rackoff. "How to construct pseudorandom permutations from pseudorandom functions." *SIAM J. of Computation*, 17(2), April 1988.

[BACKUP] Dalit Naor. Key Backup Format for Wide-block Encryption. April 14 2004

[NR98] M. Naor and O. Reingold. "A pseudo-random encryption mode." Manuscript, available from http://www.wisdom.weizmann.ac.il/~naor/.

[NR99] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1):29--66, 1999. Springer-Verlag.

[S98] R. Schroeppel. "The Hasty Pudding cipher." The first AES conference, NIST, 1998.

[ZMI89] Y. Zheng, T. Matsumoto, and H. Imai. "On the construction of block ciphers provably secure and not relying on any unproved hypotheses." In *Advances in Cryptology - CRYPTO '89*, volume 435 of Lecture Notes in Computer Science, pages 461--480. Springer-Verlag, 1989.

