Document Number: XXXX-00-0-Y-Apr04

Draft Proposal for Tweakable Narrow-block Encryption

Draft 1.00:00

August 6, 2004

Sponsor
IEEE P1619 (?)

Abstract: We describe the LRW-AES tweakable block cipher and its use for encryption of storage. LRW-AES is a tweakable block cipher that acts on "narrow" blocks of 16 bytes, and it uses as subroutine the AES block cipher (that acts on blocks of 16 bytes). Specifically, LRW-AES encrypts and decrypts blocks of 16 bytes, under the control of a secret AES *key* (that can be 16, 24, or 32 bytes), a secret 16 byte *secondary key*, and a 16-byte *tweak* generated from the secondary key and the logical position of the block. LRW-AES is a concrete instantiation of the class of tweakable block ciphers described in theorem 2 of reference [LRW02]. When used to encrypt storage data, the tweak value is computed from the logical position of the current narrow block within the scope of the current key.

The motivating application for LRW-AES is encryption of storage at the sector level. This cipher addresses threats such as copy-and-paste attacks and dictionary attacks, while allowing parallelization of cipher implementations.

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:
Clement Kent
Kasten Chase Applied Research
5100 Orbitor Drive,
Mississauga, ON, Canada
Tel: +1.905.238.6900 ext. 3237
Fax: +1.905.212.2003
Email: ckent@kastenchase.com

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 LRW-AES transform and its use for encryption of data at rest. The LRW-AES transform acts on "narrow" blocks of 16 bytes, under the control of two secret keys (the "keyset"), and the logical position of the data being encrypted within the scope of the keys (the range of data locations being encrypted). It is implemented as a mode of operation for the AES block cipher (that has blocks of size 16 bytes). The security goal of LRW-AES states that it should look like a block cipher. Moreover, using the same keyset at different positions should look like using completely independent keysets.

LRW-AES is a concrete instantiation of the class of tweakable block ciphers described in theorem 2 of reference [LRW02]. When used to encrypt storage data, the tweak value is computed from the logical position of the current narrow block within the scope of the current key.

The motivating application for LRW-AES is encryption of storage at the sector level. This cipher addresses threats such as copy-and-paste attacks and dictionary attacks, while allowing parallelization of cipher implementations. In contrast to other ciphers proposed to IEEE P1619, such as EME-32-AES (see [EME]), LRW-AES does not provide "wide" block pseudo-integrity of entire sectors. However, LRW-AES requires approximately one half the AES encrypt or decrypt operations that EME-32-AES requires.

The LRW-AES mode of operation uses a block cipher with 16-byte blocks, and turns it into a tweakable cipher with 16 byte blocks. It is proven in [LRW02] that this mode indeed achieves the stated security goal, assuming that the underlying AES 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 LRW-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". Encryption of storage data that uses blocks of different size (or variable length blocks) is expected to be out of scope for this document.

2.2 Related work

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 16, 24 or 32 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 16, 24 or 32 bytes that is used as the encryption key, and C is a block of 16 bytes. The result of this operation is a block of 16 bytes, which is denoted P.

The following symbols are used in diagrams:

⊕ Exclusive-OR operation

⊗ Multiplication of two polynomials (each of degree less than 128) modulo $x^{128}+x^7+x^2+x+1$

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 LRW-AES transform

In this section we describe the LRW-AES transform itself. That is, the procedures to be followed when encrypting or decrypting a wide block, given the secret primary key and secondary key, and the logical position of the data.

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

We now describe a procedure for multiplying a 16-byte block by *a* another 16 byte block *b* in the finite field GF(2^{128}). Both the inputs and the output of this procedure are 16-byte blocks. When these blocks are interpreted as binary polynomials of degree 127, the procedure computes $z = a \cdot b$ modulo P_{128}, where P_{128} is the polynomial $P_{128}(x) = x^{128} + x^7 + x^2 + x + 1$. Multiplication of two elements in the finite field GF(2^{128}) is computed by computing the polynomial multiplication and taking the remainder of the Euclidean division by the chosen irreducible. In our case, the irreducible is $P_{128}(x) = x^{128}+x^7+x^2+x+1$.

Example: (over GF(2^7) with irreducible x^7+x+1 for simplicity)

$\{0,1,0,0,1,1,1\} \otimes \{1,0,0,1,1,1,0\}$

$= (x^5+x^2+x+1)*(x^6+x^3+x^2+x) \bmod (x^7+x+1)$

$= (x^{11}+x^5+x^3+x) \bmod (x^7+x+1)$

$= x^4+x^3+x$

$= \{0,0,1,1,0,1,0\}$

Efficient multiplication in GF(2^{128}) may be implemented in numerous ways, depending on whether the multiplier is hardware or software based and which special optimizations may apply. Algorithm 1 represents a simple, non-optimized method:

Algorithm 1. Multiplication in GF(2^{128}). Computes the value of $Z = C \cdot Y$, where C, Y and Z ε GF(2^{128}).

$Z \leftarrow 0$, $V \leftarrow C$

for i = 0 to 127 **do**
 if Y_i = 1 **then**
  $Z \leftarrow Z \oplus V$
 end if
 if V_{127} = 0 **then**
  $V \leftarrow$ rightshift(V)
 else
  $V \leftarrow$ rightshift(V) $\oplus P_{128}$
 end if
end for
return Z

A simple optimization of this algorithm improves performance when multiple inputs Y are multiplied by the same constant C. Note that in the i^{th} step of Algorithm 1 $V = C \cdot x^i$, where x^i represents the 16 byte member of GF(2^{128}) with only the i^{th} bit set. A 2,048 byte table of V_i (i = 0 to 127) may be stored on the first use of Algorithm 1 and reused thereafter:

Algorithm 2. Multiplication in GF(2^{128}). Computes the value of $Z = C \cdot Y$, where C, Y and Z ε GF(2^{128}).

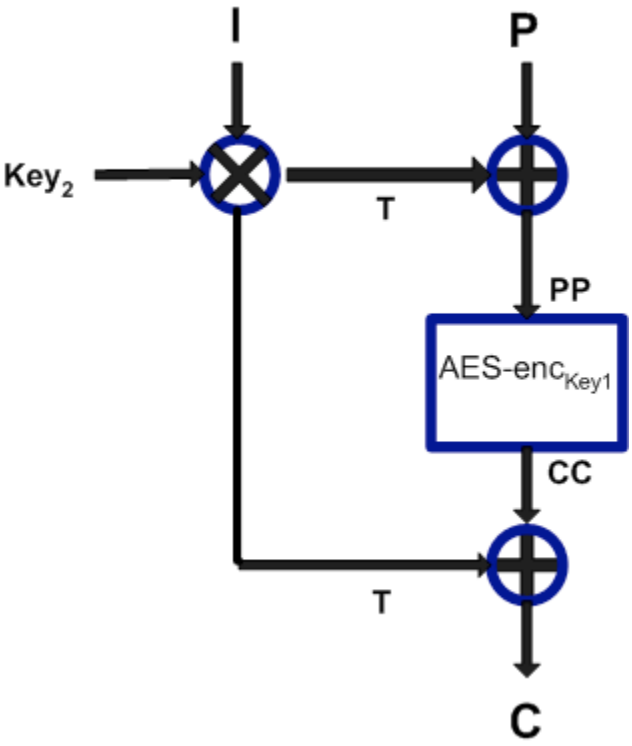
$Z \leftarrow 0$,
table of V_i (i = 0 to 127) has been precomputed

for i = 0 to 127 **do**
 if Y_i = 1 **then**
  $Z \leftarrow Z \oplus V_i$
 end if
end for
return Z

5.2 The LRW-AES encryption procedure

The LRW-AES encryption procedure takes as input a Primary AES key Key_1 (which must be an array of 16 or 32 bytes), a 16 byte Secondary key Key_2 , a 16-byte wide block ***P*** (for Plaintext) and a 16-byte block I which represents the logical position or index of the block in relation to the beginning of the key scope. It produces as output a 16-byte wide block ***C*** (for Ciphertext). Key_1 and Key_2 should be chosen according to the methods described in [R2] and Key_1 should be independent of Key_2

Figure 1. LRW-AES Encryption



Input:

P: Plaintext of length 128 bits

Key_1: Cipher Key of length 128 or 256 bits

Key_2: Tweak Key of length 128 bits

I: Logical Position Index of data within key scope in 128 bit representation

Output:

C: Ciphertext

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

1. If $I > 2^{128}-1$ or I not positive, exit and output ERROR
2. $T \leftarrow Key_2 \otimes I$
3. $PP \leftarrow P \oplus T$
4. $CC \leftarrow$ AES-enc(Key_1 , PP)
5. $C \leftarrow CC \oplus T$
6. return C

5.2.1 Optimizing calculation of *T*

An efficient implementation **may** make use of the fact that for consecutive cipher blocks, the value T can be computed by reusing previously known information as indicated below. An implementation **may** store a precomputed table of values $Key_2 \otimes X$ between LRW-AES calls.

One can easily verify that:
$T_{i+1} = Key_2 \otimes (i+1) = (Key_2 \otimes i) \oplus (Key_2 \otimes X) = T_i \oplus (Key_2 \otimes X)$
for a certain element X of the form $(\{0,0,...,0,1,1,...,1\})$

Example: (over GF(2^7) for simplicity)
$Key_2 \otimes (\{1,0,1,1,0,1,1\} + 1)$

$= Key_2 \otimes (\{1,0,1,1,1,0,0\})$

$= Key_2 \otimes (\{1,0,1,1,0,1,1\} \oplus \{0,0,0,0,1,1,1\})$

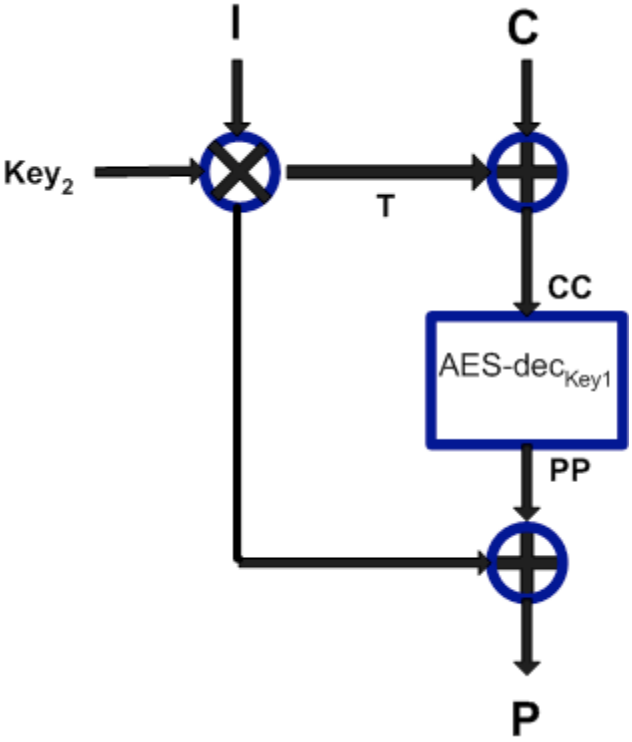
$= (Key_2 \otimes (1,0,1,1,0,1,1)) \oplus (Key_2 \otimes (0,0,0,0,1,1,1))$

As a consequence, if one precomputes $Key_2 \otimes X$ for all the possible elements X of the form $(\{0,0,...,0,1,1,...,1\})$ during the block cipher key setup for example, computation of a 512 byte sector (32 cipher blocks) will only require one field multiplication, 32 field additions and a few integer increments. Thus the incremental cost of generating T over many consecutive cipherblocks is asymptotically the cost of a table lookup and field addition (XOR).

5.3 The LRW-AES decryption procedure

The LRW-AES decryption procedure takes as input a Primary AES key Key_1 (which must be an array of 16 or 32 bytes), a 16 byte Secondary key Key_2 , a 16-byte wide block ***C*** (for Ciphertext) and a 16-byte block I which represents the logical position or index of the block in relation to the beginning of the key scope. It produces as output a 16-byte wide block ***P*** (for Plaintext).

Figure 2. LRW-AES Decryption



Input:

C: Ciphertext of length 128 bits

Key_1: Cipher Key of length 128 or 256 bits

Key_2: Tweak Key of length 128 bits

I: Logical Position Index of data within key scope in 128 bit representation

Output:

P: Plaintext of length 128 bits

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

1. If $I > 2^{128}-1$ or I not positive, exit and output ERROR
2. $T \leftarrow Key_2 \otimes I$
3. $CC \leftarrow C \oplus T$
4. $PP \leftarrow$ AES-dec(Key_1 ,CC)
5. $C \leftarrow PP \oplus T$
6. return P

6. Using LRW-AES for encryption of storage

The scope of this document is limited to direct application of the LRW-AES transform to encrypt or decrypt data at rest, when this data consists of an integral number of wide blocks. For example, a wide block may be of size 512 bytes, consisting of 32 narrow blocks, but this transform is not restricted to 512 byte wide blocks. To use this standard, an AES primary key and a 128 bit secondary key **must** 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 the primary and secondary key pair is called the *SCOPE* of that key. 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.

To encrypt a plaintext wide block with index J, the block is divided into 16-byte narrow blocks $P_{j,k}$ (k=1,N). For example, if the wide block is 512 bytes, N is 32. For each $P_{j,k}$ the logical index of the narrow block *i* is calculated as $i=k+N(J-1)$. The positive integer *i* is first encoded as a 16-byte block I, as explained in Section 6.1 below. Then the LRW-AES encryption transform is applied to this narrow block, as described in Section 5.2. The result of the transformation is the k^{th} ciphertext narrow block. Similarly, to decrypt a the k^{th} ciphertext narrow block within the wide block with index J, the positive integer logical index *i* is calculated as given above, encoded as a 16-byte block I, then the LRW-AES decryption transform is applied to this narrow block using the given key pair , as described in Section 5.3, and the result is the k^{th} plaintext narrow block of plaintext wide block with index J.

More simply stated, each narrow block of 16 bytes is associated with a logical index *i* which may be viewed as the number of 16 bytes narrow blocks from the beginning of the key scope.

In a typical application for storage encryption, the index of a wide block can be computed from the location where this wide block is stored and the scope of the key. For example, when encrypting a disk with 512-byte sectors, the scope of a key typically includes a range of logically consecutive sectors on the disk, and the index of a given sector would be its position within that sequence. In this example, the scope of a key is defined using two integers, specifying the first and last logical sectors that are associated with this key (call them X and Y). Then, a sector that is stored in logical location Z on the disk (where $X \le Z \le Y$) will have index $J = Z–X+1$ within the scope of that key.

It is stressed that an LRW-AES secret key pair **must not** be associated with more than one scope. The reason is that encrypting more than one wide block with the same key pair 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)

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

[**CW79**] J. L. Carter and M. Wegman. Universal Classes of hash functions. *Journal of Computer and System Sciences*, 18:143-154, 1979

[EME] Shai Halevi, Draft Proposal for Tweakable Wide-Block Encryption, April 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.

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

