IEEE P1619 D5

March 23, 2006

Draft Standard Architecture for Encrypted Shared Storage Media

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

Introduction

The purpose of IEEE1619 standard is to describe the method of encryption for data-at-rest in sector-based devices. In particular, the standard specifies the encryption transform and a method for exporting/importing encryption keys for compatibility between different implementations. Encryption of data-in-flight is not covered by this standard.

This standard defines the LRW-AES tweakable block cipher and its use for encryption of sector-based storage. LRW-AES is a tweakable block cipher that acts on data-units of 128 bits or more and uses AES block cipher as a subroutine. The key material for LRW-AES consists of a data encryption key (used by the AES block cipher) as well as a 16 byte "tweak key" that is used to incorporate the logical position of the data block into the encryption. LRW-AES is a concrete instantiation of the class of tweakable block ciphers described in Theorem 2 of reference [LRW02]. The LRW-AES cipher addresses threats such as copy-and-paste and dictionary attacks, while allowing parallelization and pipelining of cipher implementations.

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

James P. Hughes, *Chair*
Serge Plotkin, *Vice Chair*
Fabio Maino, *Secretary*

Shai Halevi
Dalit Naor
Rob Elliott
Eric Hibbard
Garry McCracken
Richard Moeloer
Doug Whiting
Larry Hofer
Gideon Avida

This document was edited by Serge Plotkin, Shai Halevi and Dalit Naor.

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

Contents

Introduction ii

1. Overview 5

1.1 Scope and Purpose 5

1.2 Related work 5

2. References 5

3. Definitions 6

3.1 Conformance levels 6

3.2 Glossary of terms 6

3.3 Abbreviations and acronyms 6

3.4 Numerical values 6

3.5 Letter symbols 6

4. The LRW-AES transform 7

4.1 Multiplication in the finite field GF(2^{128}) 7

4.2 The LRW-AES Encryption Procedure 7

4.2.1 LRW-AES Encryption of a Single 128-bit Block 7

4.2.2 LRW-AES Encryption of More Than 128 Bits 8

4.3 The LRW-AES Decryption Procedure 9

4.3.1 LRW-AES Decryption of a Single 128-bit Block 9

4.3.2 LRW-AES Decryption of More Than 128 Bits 10

5. Using LRW-AES for Encryption of Storage 11

5.1 Key Scope 11

5.2 Encoding the tweak values 12

6 Exporting and Archiving LRW-AES keys 12

6.1 Key Backup Structure 13

6.1.1 Structure ID 13

6.1.2 Standard 14

6.1.3 Key Scope 14

6.1.4 Transform 15
6.1.5 Key Material 15
6.1.6 Optional 15
6.2 XML Format 16
6.3 Encryption of Key Backup material 17
Annex A: Bibliography (informative) 18
Annex B: Test Vectors 19
Annex C: Rationale and Design Choices 23
C.1 Transparent Encryption 23
C.2 Wide vs. Narrow Block Tweakable Encryption 24
C.3 The Liskov-Rivest-Wagner Construction 25
C.4 Sector-size which is not a multiple of 128 bits 27
C.5 Miscellaneous 27

Standard Architecture for Encrypted Shared Storage Media

1. Overview

1.1 Scope and Purpose

The purpose of this document is to specify the LRW-AES transform, its use for encryption of data at rest, and methods for exporting/importing encryption keys for compatibility and interoperability between different implementations.

The LRW-AES transform is intended to be used to encrypt data stored on sector-level storage devices, which means that the data to be encrypted or decrypted is presented in fixed-size units, and each unit must be processed separately, independently of other data units. Another requirement for use in sector-level storage devices is that the encryption transform must be *length-preserving*, meaning that the ciphertext length must equal that of the plaintext. These two properties allow the use of LRW-AES as *transparent encryption*: an encryption/decryption module can be added to an existing system without having to modify the data layout of any of the existing components. It is noted that the requirement for transparent encryption implies some inherent limitations on the level of security that can be achieved by such a transform, and these limitations should be carefully considered by an application that uses this standard. See more discussion and rationale for LRW-AES in Annex C.

This standard includes the description of the LRW-AES transform itself (in both encryption and decryption modes) and a specification of how it should be used for encryption of data at rest. This standard also contains a description of a key-export format. The goal is to facilitate a scenario where a standard-conformant device that encrypts data using LRW-AES can export the key in a way that another standard-conformant device can later import the same key and decrypt the data.

1.2 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 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].

2. References

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

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

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

3. Definitions

3.1 Conformance levels

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

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

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

3.2 Glossary of terms

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

3.2.2 block: Sixteen bytes (=128 bits) of data.

3.2.3 key scope: Three non-negative integers representing the start and extent of the region to be encrypted by a particular key, and the data-unit size within this region.

3.2.4 key: The 256, 320, or 384 bit value that holds all of the key material used by LRW-AES transform.

3.2.5 tweak value: the 128-bit value used to represent the logical position of the data to be encrypted or decrypted with LRW-AES.

3.3 Abbreviations and acronyms

AES Advanced Encryption Standard (see NIST FIPS-197).

IEEE The Institute of Electrical and Electronics Engineers, Inc.

3.4 Numerical values

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

Decimal numbers are represented in their usual 0, 1, 2, ... format. Binary numbers are represented by a string of one or more bits followed by the subscript 2. Thus the decimal number 26 may also be represented also as 00011010_2. When describing a binary number with many repeating values, we use the notation b^n to describe the bit b repeating n times. For example, the bit pattern 100001_2 can also be written as 10^41_2.

3.5 Letter symbols

The following symbols are used in equations and figures:

$\oplus$ Bit-wise exclusive-OR operation

$\times$ Multiplication of two polynomials over the binary field GF(2).

⊗ Modular multiplication of two polynomials over the binary field GF(2), modulo $x^{128}+x^7+x^2+x+1$.

← Assignment of a value to a variable

4. The LRW-AES transform

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

The multiplication operation is defined as an operation on bit vectors in order to simplify the specification. This definition corresponds to the particular choice of the field representation used in LRW-AES. Each element is a vector of 128 bits. The *i*-th bit of an element X is denoted as X_i. The leftmost bit is X_0, and the rightmost bit is X_{127}. The multiplication operation uses the special element R = 11100001 $0^{120}{}_2$.

Following algorithm computes the value of Z = X⊗Y, where X, Y and Z are in GF(2^{128}).

```
1.  Z←0, V←X
2.  for i=0 to 127
3.     if Yi = 1
4.          Z←Z⊕V
5.      endif
6.      if V127= 0
7.          V←rightshift(V)
8.      else
9.          V←rightshift(V)⊕R
10.     endif
11. endfor
12. return Z
```

The function "`rightshift()`" moves the bits of its argument one bit to the right. More formally, whenever `W=rightshift(V)`, then $W_i=V_{i-1}$ for $1 \le i \le 127$ and $W_0 = 0$.

4.2 The LRW-AES Encryption Procedure

4.2.1 LRW-AES Encryption of a Single 128-bit Block

The LRW-AES encryption procedure for a single 128-bit block is modeled with this equation:

C = LRW-AES-blockEnc(*Key*, *P*, *i*)

where:

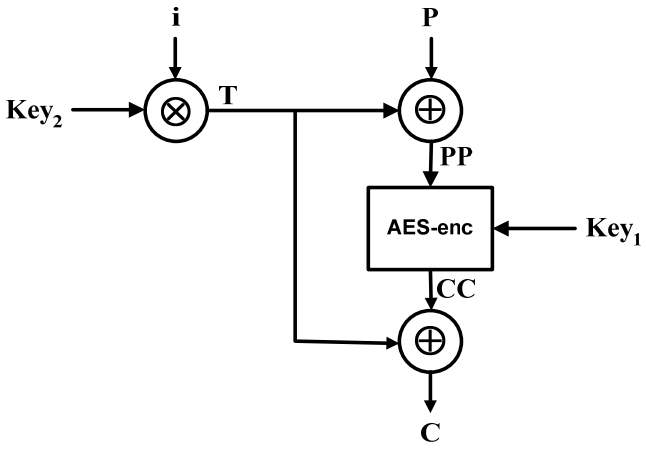
Key is the 256, 320, or 384 bit LRW-AES key
P is a block of 128 bits (i.e., the plaintext)
i is a 128-bit tweak value, representing logical position value (see section 5)
C is the block of 128 bits of ciphertext resulting from the operation

The key is parsed as a concatenation of two fields, $Key = Key_1 \mid Key_2$, with Key_2 consisting of the last 128 bits of *Key* and Key_1 consisting of the first 128, 192, or 256 bits. The ciphertext shall then be computed by the following or an equivalent sequence of steps:

1. $T \leftarrow Key_2 \otimes i$
2. $PP \leftarrow P \oplus T$
3. $CC \leftarrow$ AES-enc(Key_1, PP)
4. $C \leftarrow CC \oplus T$

AES-enc(K,P) is the procedure of encrypting plaintext P using AES algorithm with key K, according to FIPS-197.

Figure 1. LRW-AES Encryption of a single 128-bit block



4.2.2 LRW-AES Encryption of 128 or more Bits

The LRW-AES encryption procedure for a plaintext of 128 or more bits is modeled with this equation:

C = LRW-AES-Enc(*Key*, *P*, *i*),

where

Key is the 256, 320, or 384 bit LRW-AES key
P is the plaintext
i is a 128-bit integer, representing logical position value (see section 5)
C is the ciphertext resulting from the operation, of the same bit-size as *P*

The plaintext is first partitioned into *m*+1 blocks,

$P = P_0 \ldots P_{m-1}P_m$

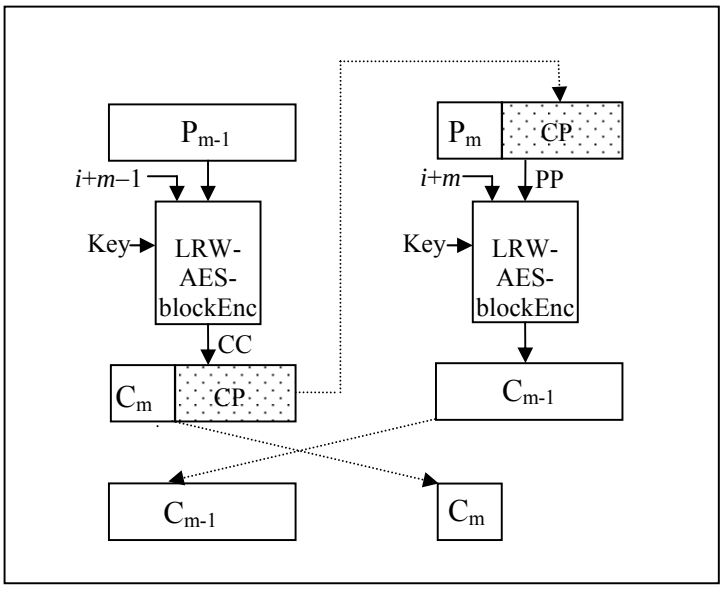
where *m* is the largest integer such that 128*m* is no more than the bit-size of P, the first *m* blocks $P_0,\ldots, P_{m-1}$ are all exactly 128-bit long, and the last block P_m is between 0 and 127-bit long. The ciphertext C is then computed by the following or an equivalent sequence of steps:

```
1.  for q=0 to m-2
2.      Cj ← LRW-AES-blockEnc(Key, Pj, i+q)       // i+q is integer addition
3.  endfor
4.  b ← bit-size of Pm
5.  if b=0
6.      Cm-1 ← LRW-AES-blockEnc(Key, Pm-1, i+m-1) // i+m-1 is integer addition
7.      Cm ← empty
8.  else                                          // Pm is a partial block
9.      CC ← LRW-AES-blockEnc(Key, Pm-1, i+m-1)  // i+m-1 is integer addition
10.     Cm ← first b bits of CC
11.     CP ← last (128-b) bits of CC
12.     PP ← Pm | CP                              // PP is a 128-bit block
13.     Cm-1 ← LRW-AES-blockEnc(Key, PP, i+m)     // i+m is integer addition
14. endif
15. C = C1 ... Cm-1 Cm
```

The result of the integer addition in the third parameter to LRW-AES-blockEnc in Lines 2, 6, 9, and 13 is viewed as a 128-bit vector with the least significant bit appearing in the leftmost position, to match the representation of the polynomial as per Section 4.1.

An illustration of encrypting the last two blocks $P_{m-1}P_m$ in the case that P_m is a partial block (b>0) is provided in Figure 2.

Figure 2. LRW-AES encryption of last two blocks when last block is smaller than 128 bits



4.3 The LRW-AES Decryption Procedure

4.3.1 LRW-AES Decryption of a Single 128-bit Block

The LRW-AES decryption procedure of a single 128-bit block is modeled with this equation:

P = LRW-AES-blockDec(*Key*, C, *i*)

where:

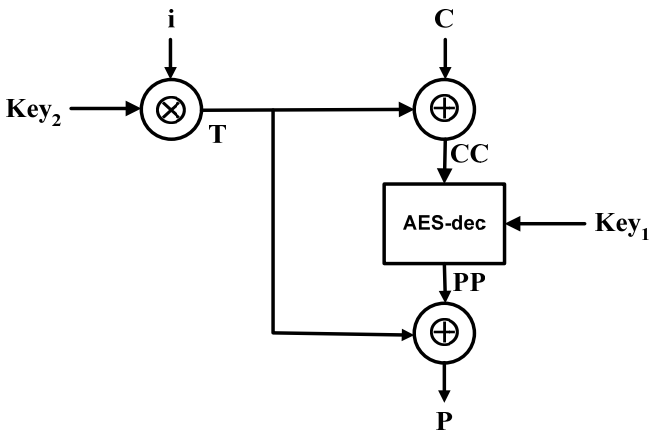
Key is the 256, 320, or 384 bit LRW-AES key
C is a block of 128 bit (i.e., the *ciphertext*)
i is a 128-bit tweak value, representing logical position value (see Section 5)
P is the 128-bit block of plaintext resulting from the operation

The key is parsed as a concatenation of two fields, $Key = Key_1 \mid Key_2$, with Key_2 consisting of the last 128 bits and Key_1 consisting of the first 256, 192, or 256 bits. The plaintext shall then be computed by the following or an equivalent sequence of steps:

1. $T \leftarrow Key_2 \otimes i$
2. $CC \leftarrow C \oplus T$
3. $PP \leftarrow$ AES-dec(Key_1 , CC)
4. $P \leftarrow PP \oplus T$

AES-dec(K,C) is the procedure of decrypting ciphertext C using AES algorithm with key K, according to FIPS-197.

Figure 3. LRW-AES Decryption of a single 128-bit block



4.3.2 LRW-AES Decryption of 128 or more Bits

The LRW-AES decryption procedure for a ciphertext of 128 or more bits is modeled with this equation:

C = LRW-AES-Dec(*Key*, *C*, *i*),

where

Key is the 256, 320, or 384 bit LRW-AES key
C is the ciphertext
i is a 128-bit integer, representing logical position value (see section 5)
P is the plaintext resulting from the operation, of the same bit-size as *C*

The ciphertext is first partitioned into m+1 blocks,

$C = C_0 \ldots C_{m-1}C_m$

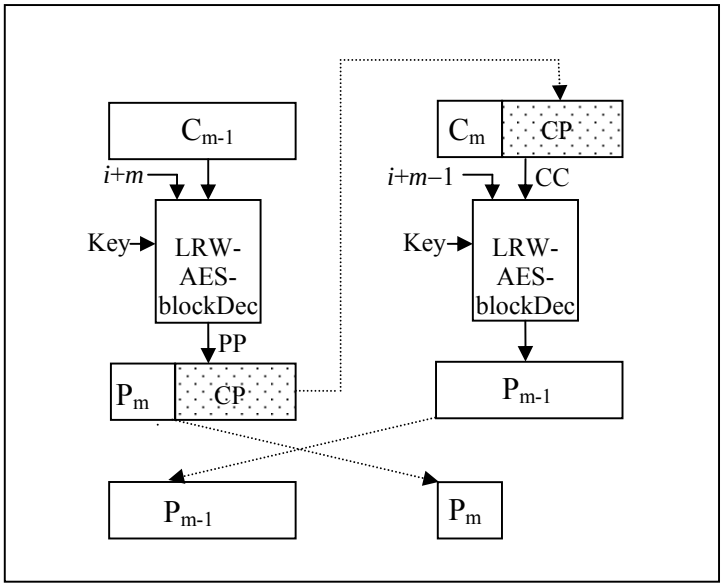
where m is the largest integer such that 128m is no more than the bit-size of C, the first m blocks $C_0,\ldots,$ C_{m-1} are all exactly 128-bit long, and the last block C_m is between 0 and 127-bit long. The plaintext P is then computed by the following or an equivalent sequence of steps:

```
1.  for q=0 to m-2
2.      Pj ← LRW-AES-blockDec(Key, Cj, i+q)        // i+q is integer addition
3.  endfor
4.  b ← bit-size of Cm
5.  if b=0
6.      Pm-1 ← LRW-AES-blockDec(Key, Cm-1, i+m-1) // i+m-1 is integer addition
7.      Pm ← empty
8.  else                                           // Cm is a partial block
9.      PP ← LRW-AES-blockDec(Key, Cm-1, i+m)      // i+m is integer addition
10.     Pm ← first b bits of PP
11.     CP ← last (128-b) bits of PP
12.     CC ← Cm | CP                               // CC is a 128-bit block
13.     Pm-1 ← LRW-AES-blockDec(Key, CC, i+m-1)    // i+m-1 is integer addition
14. endif
15. P = P1 ... Pm-1 Pm
```

The result of the integer addition in the third parameter to LRW-AES-blockDec in Lines 2, 6, 9, and 13 is viewed as a 128-bit vector with the least significant bit appearing in the leftmost position, to match the representation of the polynomial as per Section 4.1.

An illustration of the decrypting the last two blocks $C_{m-1}C_m$ in the case that C_m is a partial block (b>0) is provided in Figure 4.

Figure 4. LRW-AES decryption of last two blocks when last block is smaller than 128 bits



5. Using LRW-AES for Encryption of Storage

5.1 Key Scope

The standard defines using the LRW-AES transform to encrypt or decrypt data at rest, where the data consists of an integral number of logically contiguous data units, all of which are of the same bit length. To use this standard, a 256-bit, 320-bit, or 384-bit key shall be associated with an ordered sequence of data units, numbered consecutively. The sequence of data units that are associated with the key is related to the scope of that key. In order to encrypt or decrypt a data unit using a key, the index of that unit within the scope of the key must be known.

As defined in [6.1.3], the Key Scope is represented by three integers, the start address in bits of the storage that is 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. The data-unit size shall be at least 128 bits, and the total size of the storage that is encrypted by one key (i.e. the product of the data-unit size and the number of units) shall not exceed 2^{50} bits. A standard-conformant implementation may support multiple data-unit sizes, but this is not required.

In an application of this standard to sector-level encryption of a disk, the scope of a key typically includes a range of logically consecutive sectors on the disk, and the Start location of the Key Scope would be the location of the beginning of the first sector in the range.

An LRW-AES secret key shall not be associated with more than one Key Scope. The reason 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 attack.

5.2 Encoding the tweak values

As described in [4.2, 4.3], one of the inputs to LRW-AES encryption and decryption procedures is the value *i* that represents the logical position of the data unit (e.g., a single disk sector) that is being encrypted or decrypted. The value of *i* is determined as follows.

Let *b* be the number of bits in a single data unit, and let *j* be the ceiling of *b*/128 (i.e., *j* is the smallest integer such that $128j \geq b$). In other words, *j* is the number of 128-bit blocks that are needed to hold a single data unit. Further, let *n* be the ceiling of logarithm based 2 of *j* (i.e., *n* is smallest integer such that $2^n \geq j$). In other words, *n* is the number of bits that are needed to represent a 128-bit block position inside a data unit. Some examples of the values *b, j* and *n* for various sizes of data-units are given in Table 1 below.

Table 1 - The values of *b, j, n* for certain data-unit sizes

Size	*b*	*j*	*n*
512 bytes	4096	32	5
520 bytes	4160	33	6
528 bytes	4224	33	6
4096 bytes	32768	256	8

Let LA be the index of the current data-unit within the key-scope, with indexing starting at zero (so we have LA=0 for the first data unit within the key-scope, LA=1 for the second data-unit, etc.). The tweak value *i*, which is used to encrypt the current data unit, is computed as

$$i \leftarrow \mathrm{LA} << n,$$

where '$<< n$' denotes bit-shift by *n* positions to the left (which is equivalent to an integer multiplication by 2^n). For the sake of integer-addition (cf. Lines 2, 6, 9 and 13 in Sections 4.2.2 and 4.3.2), *i* is considered as a 128-bit integer. The result of the addition $i+q$ in these lines is then viewed as a 128-bit vector with the least significant bit appearing in the leftmost position, to match the representation of the polynomial as per Section 4.1.

6 Exporting and Archiving LRW-AES keys

The Key Backup structure provides all the information that is needed in order to decrypt an integral number of data units that were encrypted with LRW-AES. The sequence of data units that are associated with a specific Key Backup structure (in particular, a specific key) is called the Key Scope of the Key Backup structure.

6.1 Key Backup Structure

Table 2 lists the elements of the key backup structure.

Table 2 – Key Backup Structure

Field	Format	Description
StructureID	Table 3 – Structure ID	Identifier of current structure
Standard	Table 4 – Standard Description	Standard identifier
KeyScope	Table 5 – Key Scope	Key Scope
Transform	Table 6 – Transform Description	Transform description
KeyMaterial	Table 8 – Key Material	Key material and its length
Optional	Table 9 – Optional Parameters	Additional unspecified information

6.1.1 Structure ID

Table 3 defines the StructureID element, which is the information needed to uniquely identify a particular instance of a key backup structure.

Table 3 – Structure ID

Field	Size	XML Encoding	Description
ID	16 bytes	Base64	General identifier for the key backup structure.
Comment	Up to 1024 bytes	Text	A text description provided by the vendor.

6.1.2 Standard

This element is used to uniquely define standard-related information relevant to the time of encryption. The standard is expected to evolve and extend over time. Table 4 defines the Standard element.

Table 4 – Standard Description

Field	Size	XML Encoding	Description
StandardNumber	Up to 128 bytes	Text	Number of the Standard used. Specified in the format xxxx.yyy
StandardVersion	2 bytes	Integer	Specifies the version of the Standard Number used
StandardComment	Up to 256 bytes	Text	Any additional standard-related information.

6.1.3 Key Scope

The KeyScope specifies the scope of the key material that is identified in the key backup structure. The scope is an ordered sequence of data-units, numbered consecutively starting at a certain position. Table 5 defines the KeyScope element.

Table 5 – Key Scope

Field	Size	XML Encoding	Description
KeyScopeStart	16 bytes	Integer	The number of bits from the start of the medium (e.g., from logical block address 0) until the beginning of the data that is encrypted with the current key.
DataUnitSize	16 bytes	Integer	The number of bits in one data-unit that is covered by the current key.
KeyScopeLength	16 bytes	Integer	The number of data-units that are covered by the current key.

6.1.4 Transform

Table 6 defines the Transform element.

Table 6 – Transform Description

Field	Size	XML Encoding	Description
TransformName	Up to 16 bytes	Text	The transform name.

The transform name shall be one of the supported strings, as specified in Table 7 below.

Table 7 – Supported Transforms

String	Description
LRW-AES	The LRW-AES transform as defined in this standard.

6.1.5 Key Material

Table 8 defines the KeyMaterial element. All key material is secret.

Table 8 – Key Material

Field	Size	XML Encoding	Description
KeyLength	2 bytes	Integer	Length (in bits) of the key. Allowed values are 256, 320, 384
KeyValue	variable	Base64	The value of the key.

6.1.6 Optional

Table 9 defines the OptionalParameters element.

Table 9 – Optional Parameters

Field	Size	XML Encoding	Description
OptionalParameters	Variable	Text	Additional parameters to the key-backup structure, in (Attribute, Value) format. These parameters can either be defined by the standard or be user-defined or vendor-specific attributes.

6.2 XML Format

The Key Backup structure is encoded in XML, to facilitate a unified format and allow an application independent way of sharing key material. This also provides an automatic generation and parsing of Key backup structures. Following is a DTD (Document Type Definition) for the P1619 Key Backup format:

```
<!ELEMENT KeyBackup (StructureID, Standard, KeyScope, Transform, KeyMaterial,
OptionalParameters)>
   <!ELEMENT StructureID  (ID, Comment?)>
   <!ELEMENT ID (#PCDATA)>
      <!ATTLIST ID Encoding CDATA #FIXED "Base64">
   <!ELEMENT Comment      (#PCDATA)>
   <!ELEMENT Standard     (StandardNumber, StandardVersion, StandardComment?)>
   <!ELEMENT StandardNumber   (#PCDATA)>
   <!ELEMENT StandardVersion  (#PCDATA)>
      <!ATTLIST StandardVersion Encoding CDATA #FIXED "Integer">
   <!ELEMENT StandardComment  (#PCDATA)>
   <!ELEMENT KeyScope (KeyScopeStart, DataUnitSize, KeyScopeLength)>
   <!ELEMENT KeyScopeStart    (#PCDATA)>
      <!ATTLIST KeyScopeStart Encoding CDATA #FIXED "Integer">
   <!ELEMENT DataUnitSize     (#PCDATA)>
      <!ATTLIST DataUnitSize Encoding CDATA #FIXED "Integer">
   <!ELEMENT KeyScopeLength   (#PCDATA)>
      <!ATTLIST KeyScopeLength Encoding CDATA #FIXED "Integer">
   <!ELEMENT Transform        (TransformName)>
   <!ELEMENT TransformName    (#PCDATA)>
   <!ELEMENT KeyMaterial  (KeyLength, KeyValue)>
   <!ELEMENT KeyLength        (#PCDATA)>
      <!ATTLIST KeyLength Encoding CDATA #FIXED "Integer">
   <!ELEMENT KeyValue         (#PCDATA)>
      <!ATTLIST KeyValue Encoding CDATA #FIXED "Base64">
   <!ELEMENT OptionalParameters (#PCDATA)>
```

An example of an XML document containing a single key:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE KeyBackup SYSTEM "keybackup.dtd">
<KeyBackup>
<StructureID>
   <ID Encoding="Base64">A92A01183H5G</ID>
   <Comment>Comment text here</Comment>
   </StructureID>
<Standard>
   <StandardNumber>IEEE P1619</StandardNumber>
   <StandardVersion Encoding="Integer">3</StandardVersion>
   <StandardComment>Disk</StandardComment>
   </Standard>
<KeyScope>
   <KeyScopeStart Encoding="Integer">0</KeyScopeStart>
   <DataUnitSize Encoding="Integer">512</DataUnitSize>
   <KeyScopeLength Encoding="Integer">1083</KeyScopeLength>
   </KeyScope>
<Transform>
   <TransformName>LRW-AES</TransformName>
   </Transform>
<KeyMaterial>
   <KeyLength Encoding="Integer">384</KeyLength>
   <KeyValue Encoding="Base64">AAAAAAAA....</KeyValue>
</KeyMaterial>
```

```
<OptionalParameters>KeyName="LUN 7 on XYZ" </OptionalParameters>
</KeyBackup>
```

6.3 Encryption of Key Backup material

A Key backup structure may be protected as follows.

1. The actual key material (KeyMaterial from Table 2) shall be wrapped encrypted with xml-enc [XML-ENC] and embedded within the XML key backup structure. [XML-ENC] does not mandate any single key wrapping algorithm, but to be compliant with this standard, vendors shall support NIST AES 256 Key Wrap (see http://www.w3.org/2001/04/xmlenc#kw-aes256). Other key wrap algorithms allowed by [XML-ENC] may be used.
2. The keys that are used to wrap the key elements (KEK) may be referenced with xkms [XML-KMS]. The location of wrapping keys is not specified by this standard. The cryptographic strength of wrapping keys should be at least equivalent to the strength of the storage encryption keys wrapped (see reference [KEY-MGMT]).
3. Implementation shall provide integrity for the key file, using standard methods.

An example of the XML encoding of the KeyMaterial fields is:

```
<KeyMaterial>
    <EncryptedKey>Type='http://www.w3.org/2001/04/xmlenc#Content'
        xmlns="http://www.w3.org/2001/04/xmlenc#"
    <EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes256"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>WrapKey</ds:KeyName>
    </ds:KeyInfo>
    <CipherData><CipherValue>B457V645B45........</CipherValue>
    </CipherData>
    </EncryptedKey>
</KeyMaterial>
```

In the above example, the KeyMaterial is encrypted using AES-256 Key Wrap, whose wrapping key has identifier “WrapKey”.

An alternative format allowed by [XML-ENC] would use a wrapping key reference, rather than a wrapping key name:

```
<KeyMaterial>
  <EncryptedKey>
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes256"/>
    <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
         <ds:RetrievalMethod URI='#WK'/>
         </ds:KeyInfo>
    <CipherData>
        <CipherValue>B457V645B45........</CipherValue>
    </CipherData>
  </EncryptedKey>
</KeyMaterial>
```

If a wrapping key reference is used as above the wrapping key would be stored as an EncryptedKey element elsewhere in the document with ID=’WK’.

Annex A: Bibliography (informative)

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

[MM82] C.H. Mayer and S. M. Matyas, "Cryptography: a New Dimension in Computer Security." John Wiley & Sons, 1982.

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

[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.

[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.

[H04] S. Halevi. "EME*: extending EME to handle arbitrary-length messages with associated data," INDOCRYPT 2004, LNCS vol. 3348, pages 315-327. Springer-Verlag, 2004.

[**KEY-MGMT]** NIST Key Management Guideline. http://csrc.nist.gov/CryptoToolkit/kms/key-management-guideline-(workshop).pdf

Annex B: Test Vectors

This Annex lists several examples of applying LRW-AES. All numbers in this Annex are hexadecimal. For readability, the examples explicitly parse the LRW-AES key into Key_1 and Key_2.

LRW-AES with 32-byte key material applied to 16-byte cleartext

```
Key1        4562ac25f828176d4c268414b5680185
Key2        258e2a05e73e9d03ee5a830ccc094c87
I           80000000000000000000000000000000
P           30313233343536373839414243444546
T           258e2a05e73e9d03ee5a830ccc094c87
PP          15bf1836d30bab34d663c24e8f4d09c1
CC          d43c59c8829d425c0707cb9e986a023f
C           f1b273cd65a3df5fe95d489254634eb8

Key1        59704714f557478cd779e80f54887944
Key2        3523c2dec5694fa872a9aca70b2beebc
I           40000000000000000000000000000000
P           30313233343536373839414243444546
T           1a91e16f62b4a7d43954d6538595f75e
PP          2aa0d35c568191e3016d9711c6d1b218
CC          1a59cac1f70f6a311e1bd13a37f51668
C           00c82bae95bbcde5274f0769b260e136

Key1        59704714f557478cd779e80f54887944
Key2        6753c90cb7d8cde506a047781aad8511
I           00000000000000000000000000000002
P           30313233343536373839414243444546
T           1a91e16f62b4a7d43954d6538595f75e
PP          2aa0d35c568191e3016d9711c6d1b218
CC          1a59cac1f70f6a311e1bd13a37f51668
C           00c82bae95bbcde5274f0769b260e136

Key1        d82a9134b26a565030fe69e2377f9847
Key2        4eb55d3105973a3f5e23dafb5a45d6c0
I           00000000000000000000000200000000
P           30313233343536373839414243444546
T           18c91f6d601a1a375d0b0ef73ad574c4
PP          28f82d5e542f2c0065324fb579913182
CC          6efb3eee8d95ebb5a4526cf453db2ac5
C           76322183ed8ff182f9596203690e5e01
```

LRW-AES with 32-byte key material applied to 20-byte cleartext

```
Key1        d82a9134b26a565030fe69e2377f9847
Key2        4eb55d3105973a3f5e23dafb5a45d6c0
I           00000000000000000000000200000000
P           3031323334353637383941424344454665666768

I0:         00000000000000000000000200000000
T0:         18c91f6d601a1a375d0b0ef73ad574c4
P0:         30313233343536373839414243444546
PP0:        28f82d5e542f2c0065324fb579913182
CC0:        6efb3eee8d95ebb5a4526cf453db2ac5
(C1|CP):    76322183ed8ff182f9596203690e5e01
C1:         76322183
I1:         00000000000000000000000200000001
T1:         ed7683d2287e9581ed0603ec9450f3bf
P1:         65666768
```

```
(P1|CP):  65666768ed8ff182f9596203690e5e01
PP1:      8810e4bac5f16403145f61effd5eadbe
CC1:      93ba4fdf31c057fcc74f969e56921f91
C0:       7ecccc0d19bec27d2a499572c2c2ec2e

C         7ecccc0d19bec27d2a499572c2c2ec2e76322183
```

LRW-AES with 32-byte key material applied to 32-byte cleartext

```
Key1      d82a9134b26a565030fe69e2377f9847
Key2      4eb55d3105973a3f5e23dafb5a45d6c0
I         00000000000000000000000200000000
P         3031323334353637383941424344454630313233343536373839414243444546

I0:       00000000000000000000000200000000
T0:       18c91f6d601a1a375d0b0ef73ad574c4
P0:       3031323334353637383941424344454 6
PP0:      28f82d5e542f2c0065324fb579913182
CC0:      6efb3eee8d95ebb5a4526cf453db2ac5
C0:       76322183ed8ff182f9596203690e5e01

I1:       00000000000000000000000200000001
T1:       ed7683d2287e9581ed0603ec9450f3bf
P1:       3031323334353637383941424344454 6
PP1:      dd47b1e11c4ba3b6d53f42aed714b6f9
CC1:      bede258be1bee410e0563d5f607a1533
C1:       53a8a659c9c071910d503eb3f42ae68c

C         76322183ed8ff182f9596203690e5e0153a8a659c9c071910d503eb3f42ae68c
```

LRW-AES with 32-byte key material applied to 33-byte cleartext

```
Key1      d82a9134b26a565030fe69e2377f9847
Key2      4eb55d3105973a3f5e23dafb5a45d6c0
I         00000000000000000000000200000000
P         3031323334353637383941424344454630313233343536373839414243444546 65

I0:       00000000000000000000000200000000
T0:       18c91f6d601a1a375d0b0ef73ad574c4
P0:       3031323334353637383941424344454 6
PP0:      28f82d5e542f2c0065324fb579913182
CC0:      6efb3eee8d95ebb5a4526cf453db2ac5
C0:       76322183ed8ff182f9596203690e5e01

I1:       00000000000000000000000200000001
T1:       ed7683d2287e9581ed0603ec9450f3bf
P1:       3031323334353637383941424344454 6
PP1:      dd47b1e11c4ba3b6d53f42aed714b6f9
CC1:      bede258be1bee410e0563d5f607a1533
(C2|CP):  53a8a659c9c071910d503eb3f42ae68c
C2:       53

I2:       00000000000000000000000200000002
T2:       31b62613f0d3055a3d1114c067de7a33
P2:       65
(P2|CP):  65a8a659c9c071910d503eb3f42ae68c
PP2:      541e804a391374cb30412a7393f49cbf
CC2:      17dc0f892ee908d186a11e35aaee2dc2
C1:       266a299ade3a0d8bbbb00af5cd3057f1
```

```
C          76322183ed8ff182f9596203690e5e01266a299ade3a0d8bbbb00af5cd3057f153
```

LRW-AES with 40-byte key material applied to 16-byte cleartext

```
Key1       0f6aeff8d3d2bb152583f73c1f012874cac6bc354d4a6554
Key2       90ae61cf7baebdccade494c54a29ae70
I          80000000000000000000000000000000
P          30313233343536373839414243444546
T          90ae61cf7baebdccade494c54a29ae70
PP         a09f53fc4f9b8bfb95ddd587096deb36
CC         0ca174e02e0c653c7b9f1b5b620b1231
C          9c0f152f55a2d8f0d67b8f9e2822bc41

Key1       0f6aeff8d3d2bb152583f73c1f012874cac6bc354d4a6554
Key2       20c54e9572f5468f0d3b4bf1de3b8020
I          00000000000000000000000000000001
P          30313233343536373839414243444546
T          90ae61cf7baebdccade494c54a29ae70
PP         a09f53fc4f9b8bfb95ddd587096deb36
CC         0ca174e02e0c653c7b9f1b5b620b1231
C          9c0f152f55a2d8f0d67b8f9e2822bc41
```

LRW-AES with 48-byte key material applied to 20-byte cleartext

```
Key1       fb7615b23d80891dd470980bc79584c8b2fb64ce6097878d17fce45a49e830b7
Key2       85b1ca1a9e1995da06fffcb016220f6f
I          00000000000000000000000200000000
P          303132333435363738394142434445466656666768

I0:        00000000000000000000000200000000
T0:        5abc25a8c0c808f5e25f3c746ec7286a
P0:        30313233343536373839414243444546
PP0:       6a8d179bf4fd3ec2da667d362d836d2c
CC0:       012cab696b156faadf36b6e13d0fb48f
(C1|CP):   5b908ec1abdd675f3d698a9553c89ce5
C1:        5b908ec1

I1:        00000000000000000000000200000001
T1:        7bd859a247961a21823b380e9fe8b650
P1:        65666768
(P1|CP):   65666768abdd675f3d698a9553c89ce5
PP1:       1ebe3ecaec4b7d7ebf52b29bcc202ab5
CC1:       a3ee2d585ca7954b64d7d0cfea1e4859
C0:        d83674fa1b318f6ae6ece8c175f6fe09

C          d83674fa1b318f6ae6ece8c175f6fe095b908ec1
```

LRW-AES with 40-byte key material applied to 16-byte cleartext

```
Key1        8ad4ee102fbd81fff886ceac93c5adc6a01907c09df7bbdd
Key2        dbd1a3636024b7b402da7d7de61d4180
I           00000000000000000000000200000000
P           30313233343536373839414243444546
T           e1fe23b1ac11a19a5d622e8f6f468d8d
PP          d1cf1182982497ad655b6fcd2c02c8cb
CC          35d949ceb8809cff9502668de8a5b98b
C           d4276a7f14913d65c860480287e33406
```

LRW-AES with 48-byte key material applied to 16-byte cleartext

```
Key1        f8d476ffd646ee6c2384cb1c77d6195dfef1a9f37bbc8d21a79c21f8cb900289
Key2        a845348ec8c5b5f126f50e76fefd1b1e
I           80000000000000000000000000000000
P           30313233343536373839414243444546
T           a845348ec8c5b5f126f50e76fefd1b1e
PP          987406bdfcf083c61ecc4f34bdb95e58
CC          15438c6f135d3c6fe26deae731e16b35
C           bd06b8e1db98899ec498e491cf1c702b

Key1        f8d476ffd646ee6c2384cb1c77d6195dfef1a9f37bbc8d21a79c21f8cb900289
Key2        c0b499f0dca213c9b6c4098deb8cc6d8
I           00000000000000000000000000000001
P           30313233343536373839414243444546
T           a845348ec8c5b5f126f50e76fefd1b1e
PP          987406bdfcf083c61ecc4f34bdb95e58
CC          15438c6f135d3c6fe26deae731e16b35
C           bd06b8e1db98899ec498e491cf1c702b

Key1        fb7615b23d80891dd470980bc79584c8b2fb64ce6097878d17fce45a49e830b7
Key 2       85b1ca1a9e1995da06fffcb016220f6f
I           00000000000000000000000200000000
P           30313233343536373839414243444546
T           5abc25a8c0c808f5e25f3c746ec7286a
PP          6a8d179bf4fd3ec2da667d362d836d2c
CC          012cab696b156faadf36b6e13d0fb48f
C           5b908ec1abdd675f3d698a9553c89ce5
```

Annex C: Rationale and Design Choices

In this section we provide some background material regarding some design choices that were made in LRW-AES and the rationale behind these choices.

C.1 Transparent Encryption

The starting point for this standard is a requirement that the transform be usable as *transparent encryption.* That is, it should be possible to insert an encryption/decryption module into existing data paths without having to change the other components on these data paths. In particular, transparent encryption can be implemented to occur in the host, along the data path from host to storage device, and inside the storage device, all without the need to modify the data transmission protocols or the layout of the data on the media. In the context of encryption by "sector-level storage devices" (as specified in the PAR document for this standard), this requirement translates into the following two constraints:

1. The transform must be *length-preserving*, namely the length of the ciphertext must equal that of the plaintext. This means that the transform must be deterministic, and that it cannot store an authentication tag along with the ciphertext.

2. The transform must be applicable to individual data-units (or sectors) independently of other data-units and in arbitrary order. This means that no chaining between different data-units is possible. This requirement stems from the need to support random access to the encrypted data. For example, encryption mode that chains multiple data units requires reading of several data units to decrypt a single unit.

Two solutions that were rejected by the group as insecure were to use either counter mode or CBC mode, deriving the IV from the sector number.

- Using counter-mode without authentication tags is trivially malleable, and an attacker with write access to the encrypted media can flip any bit of the plaintext simply by flipping the corresponding ciphertext bit.

- The reason why CBC (even with position-based IV) is not sufficiently secure is that an attacker with read/write access to the encrypted disk can copy a ciphertext sector from one position to another, and an application reading the sector off the new location will still get the same plaintext sector (except perhaps the first 128 bits). For example, this means that an attacker that is allowed to read a sector from the second position but not the first can find the content of the sector in first position by manipulating the ciphertext.

- Another drawback of CBC mode is that an attacker can flip any bit in the plaintext, if it is willing to pay the price of randomizing the previous plaintext block.

As described below, the LRW-AES transform was chosen because it offers better protection against ciphertext manipulations and cut-and-paste attacks. It is important to realize, however, that regardless of the method used for encryption, the constraints above imply some inherent limitations on the level of security that can be achieved by such transform. As we explain below, these constraints imply that the best achievable security is essentially what can be obtained by using ECB mode with a different key per block (and using a cipher with wide blocks).

Specifically, since there are no authentication tags then any ciphertext (original or modified by attacker) will be decrypted as some cleartext and there is no built-in mechanism to detect alternations. The best that can be done is to ensure that any alternation of the ciphertext will completely randomize the plaintext, and

rely on the application that uses this transform to include sufficient redundancy in its plaintext to detect and discard such random plaintexts.

Also, since this transform is deterministic, then encrypting the plaintext twice with the key and the same position will necessarily yield the same ciphertext. Moreover, since there is no chaining then an attacker can "mix and match" ciphertext units and get the same "mix and match" of their corresponding plaintext units. (Namely, if $C_1C_2\ldots C_m$ is encryption of $P_1P_2\ldots P_m$ and $C'_1C'_2\ldots C'_m$ is encryption of $P'_1P'_2\ldots P'_m$ then $C_1C'_2\ldots C_m$ is encryption of $P_1P'_2\ldots P_m$.)

The last limitations can be mitigated to some extent by using some context information in the encryption and decryption processes. In the case of sector-level encryption, the only context information that can be assumed to be available at both encryption and decryption is the (logical) position of the current data unit (as seen by the encryption/decryption module).[2] Incorporating the position information into the encryption and decryption routines makes it possible to cryptographically hide the fact that the same unit is written in two different places, and also prevents "mix and match" between different positions. But as mentioned above, even the best implementation of encryption by a sector-level storage device leaves some vulnerabilities. Three of these vulnerabilities are illustrated next.

- **Traffic analysis.** Consider an attacker that is able to passively observe the communication between the encrypting device and the disk. Since encryption is deterministic, this attacker is able to observe when a certain sector is written back to disk with a different value than was previously read from disk. This capability may help the attacker in mounting an attack based on traffic analysis.

- **Replay.** An attacker with read/write access to the encrypted disk can observe when a certain sector changes on the disk and then reset it to its previous value. (Notice that this attack is not specific to transparent encryption, it may work even when using randomized encryption with authentication tags.)

- **Randomizing a sector.** Since there are no authentication tags, an attacker with write access to the encrypted disk can write an arbitrary ciphertext to any sector, causing an application that reads this sector to see a "random" plaintext instead of the value that was written to that sector. The behavior of the application on such "random" plaintext may be beneficial to the attacker.

When using encryption at the sector level, one must therefore address these vulnerabilities by other means.

C.2 Wide vs. Narrow Block Tweakable Encryption

In light of the discussion above, the interfaces of the transform that we seek are encryption and decryption routines:

$$C = \mathrm{Enc}(K, P, i) \text{ and } P = \mathrm{Dec}(K, C, i),$$

where the plaintext P and ciphertext C have the same length (i.e., the length of a single sector), K is the secret encryption key, and *i* represents the position information.

The best security that one can hope for with such transform is that it looks to an attacker like a block cipher with block size equal to the sector size, and with different and independent keys for different values of *i*. (Such a construct is called a "tweakable cipher" in the cryptographic literature. It was first defined formally by Liskov et al. in [LRW02].)

[2] On the other hand, parameters like "time of encryption" cannot be used as context information, since the decryption procedure typically has no way of obtaining that information.

Several constructions that achieve these properties exist in the cryptographic literature (e.g., [HR03], [HR04], [H04], and a construction based on [NR98], just to name a few). All these constructions, however, are rather expensive: they all induce at least as much work as two passes over the entire sector with the underlying block cipher, and they all require buffering of at least one sector worth of intermediate results.[3] A cheaper alternative can be obtained by relaxing the requirement that the transform looks like a cipher with a wide (e.g. sector-length) block-size. Instead, we work with narrow blocks of 128 bits, but still insist that different blocks (whether in the same or in different sectors) look to an attacker like they were encrypted with different independent keys.

As we discuss below, giving up the dependencies between different 128-bit blocks allows greater efficiency. The price for that, however, is that the attacks described in C.1 are now possible with better granularity. Namely, whereas the attacker against a wide-block encryption scheme can do traffic analysis or replay with granularity of one sector, the attacker against a narrow-block encryption scheme can work with granularity of 128-bit blocks. Still, the consensus in the group was the added efficiency warrants this additional risk. (Since these risks exist even with wide-block encryption – albeit with a coarser granularity – then one would anyway need some other mechanisms for addressing them, and in many cases the same mechanisms can be used also for addressing these risks in their fine-grained form.)

C.3 The Liskov-Rivest-Wagner Construction

Liskov et al. described in [LRW02] two general constructions for narrow-block tweakable ciphers from a standard cipher such as AES. The second construction from that paper works as follows: The tweakable cipher uses two keys: one key K_1 is used with the underlying cipher ENC_{K1}/DEC_{K1}, and the other key K_2 is used with a "non-cryptographic hash function" H_{K2} to process the tweak value. Given a plaintext block P and the tweak value *i*, the construction first applies the hash function to *i* to get $T \leftarrow H_{K2}(i)$ and then uses the underlying block cipher to compute $C \leftarrow ENC_{K1}(P \oplus T) \oplus T$. (To decrypt, it computes $T \leftarrow H_{K2}(i)$ and then sets $P \leftarrow DEC_{K1}(C \oplus T) \oplus T$.)

Security Analysis. The property of the hash function that is needed for the security of this construction is that for any two fixed different tweak values $i \neq i'$, choosing the second key K_2 at random and computing $\Delta \leftarrow H_{K2}(i) \oplus H_{K2}(i')$ induces the uniform distribution on Δ (i.e., each of the 2^n possible values for Δ occurs with probability 2^n, where *n* is the block-size of the underlying cipher).

The attacker analyzed in [LRW02] can make arbitrary encryption and decryption queries to the tweakable cipher (using arbitrary tweak values). These queries are answered either by the construction above, or by a truly random collection of permutations and their inverses over $\{0,1\}^n$ (a different, independent permutation for every value of the tweak *i*), and the attackers goal is to determine which is the case. Liskov et al. proved in [LRW02, Theorem 2] that if the hash function has this property, then an attacker that makes at most *q* such queries cannot distinguish these two cases with advantage more than $3q^2/2^n + \delta$ over a random guess (where δ is an error terms that expresses the advantage of distinguishing the underlying cipher from a random permutation using *q* queries).

At this point, it is worth explaining the relevance of this analysis to the security of a real-world usage of the LRW-AES transform. First, we argue that no realistic attacker would have more information than the attacker in the attack model that is described in the analysis (since that attacker can choose all the plaintext and ciphertext that is fed to the construction). And since the theorem says that no attacker in that model can distinguish the construction from a collection of random permutations, it follows that no realistic attacker can distinguish between these cases with any significant advantage. This, in turn, means that whatever attack we are facing would be just as successful if we were using a collection of truly random permutations, one per each 128-bit block, to encrypt our data rather than using the scheme of Liskov et al.

[3] At least some of this overhead appears to be inherent: Since these schemes insist on a block cipher with "wide block" (i.e., as wide as an entire sector), then every bit of ciphertext must "strongly depend" on every bit of plaintext and vice versa. This means in particular that no bit of output can be produced until all the input bits were processed by the block cipher.

It follows that when analyzing the security of an application that uses that scheme, we can think of the encryption as if it was done using a collection of truly random permutations. Notice now that when faced with such a collection of truly random permutations, the only information that the attacker has is the following:

- The same plaintext with the same tweak value will always be encrypted to the same ciphertext (cf. the traffic analysis attack from above).
- The same ciphertext with the same tweak value will always be decrypted to the same plaintext (cf. the replay attack from above).
- Any other ciphertext (resp. plaintext) will be decrypted (resp. encrypted) to a random value (cf. the randomizing attack from above).

Hence, the theorem from [LRW02] essentially tells us that except for the "error term" of $3q^2/2^n + \delta$, the only attacks that are possible against this scheme are the ones that are inherent from the use of transparent encryption (with the granularity of *n*-bit blocks).

LRW-AES. The LRW-AES transform that is described in this standard is a concrete instantiation of the Liskov-Rivest-Wagner scheme, with AES as the underlying block cipher and the hash function computed as $H_{K2}(i) = K_2 \otimes i$ (with $\otimes$ a multiplication in the finite field $GF(2^{128})$). It is easy to see that for any fixed $i \neq i'$ we have

$$H_{K2}(i) \oplus H_{K2}(i') = (K_2 \otimes i) \oplus (K_2 \otimes i') = K_2 \otimes (i \oplus i').$$

Since $i \neq i'$ then $i \oplus i' \neq 0$ and therefore $K_2 \otimes (i \oplus i')$ is uniform in $GF(2^{128})$ when K_2 is. Thus, the hash function satisfies the condition that is needed for Theorem 2 in [LRW02] and we can use that theorem to argue about the security of LRW-AES.

When using AES as the underlying block cipher, we have n=128 and the expression $3q^2/2^n$ would be small enough as long as q is not much more than 2^{40}. This means that practically speaking there should be no problem using the same key to encrypt a terabyte of data (which gives $q=2^{36}$ blocks), but one probably should not be using the same key for a petabyte of data (2^{46} blocks).[4]

Efficient Implementation of LRW-AES. The way LRW-AES is defined, it requires calculation of a multiplication in $GF(2^{128})$ per each 128 bits of output. Since this multiplication might be expensive, it is sometimes useful to optimize it. One possible optimization uses the fact that when processing a sector with several consecutive 128-bit blocks, the tweak values that are used for these blocks are i, i+1, …, i+j−1 where i is a multiple of 2^n and $j \leq 2^n$). Once we compute the initial value of $K_2 \otimes i$, we can use the following optimization for quickly computing $K_2 \otimes (q+1)$ from $K_2 \otimes q$ (for q between i and i+j−2). We observe that

$$K_2 \otimes q \oplus K_2 \otimes (q+1) = K_2 \otimes (q \oplus (q+1))$$

Thus, if $K_2 \otimes q$ is known, instead of computing $K_2 \otimes (q+1)$ directly, it is sufficient to compute:

$$K_2 \otimes (q+1) = (K_2 \otimes q) \oplus (K_2 \otimes (q \oplus (q+1)))$$

The conditions stated above (i is a multiple of 2^n and $i \leq q < i+2^n-1$) imply that the term $(q \oplus (q+1))$ can only assume one of the n values 1, 3, 7, 15, …, 2^n-1. It is therefore possible to pre-compute a table with these n values of $K_2 \otimes (2^s-1)$. Given such table, computing $K_2 \otimes (q+1)$ from $K_2 \otimes q$ can be done by

[4] For these values of q it seems reasonable to assume that the error term δ that comes from attacking AES is insignificant.

computing $(q \oplus (q+1))$, using the result to lookup $K_2 \otimes (q \oplus (q+1))$ and xor-ing it with $K_2 \otimes q$. This way, computing LRW-AES on a *j*-block sector is only slightly more expensive than encrypting the same sector using ECB mode.

Note that the procedure above is just an optimization example. There is no requirement to follow it to be compatible with the standard.

C.4 Sector-size which is not a multiple of 128 bits

The LRW-AES transform as described above immediately implies a method for encrypting sectors that consist of an integral number of 128-bit blocks: apply LRW-AES individually to each 128-bit block, but use the block number in the sector as part of the tweak value when encrypting that block. This method is applicable to the most common sector sizes (such as 512 bytes or 4096 bytes). However, it does not directly apply to sector sizes that are not an integer multiple of 128-bit blocks (e.g., 520-byte sectors as in AS400 systems).

To encrypt sectors that are not an integral number of 128-bit blocks, the standard uses the "ciphertext-stealing" technique similar to the one used for ECB mode (cf. [MM82, Fig. 2-22]). Namely, LRW-AES encrypts all the full blocks *except the last* (with different tweak values), and then encrypts the last full block together with the remaining partial block using two application of the 128-bit LRW-AES-blockEnc procedure described in Section 4.2.1 with two different tweak values, as described in Figure 2 in Section 4.2.2

C.5 Miscellaneous

We conclude this section with some (rather obvious) remarks about "appropriate use" of the LRW-AES transform.

- When analyzing the security of an application that uses this standard, one must consider the methods that were used to generate the keys. As with every cryptographic algorithm, it is important that the secret-key used for LRW-AES be chosen at random (or from a "cryptographically strong" pseudo-random source). Indeed, all security guarantees (including the security claims of the theorem from [LRW02]) are null and void if the key is chosen from a low entropy source. The issues of strong pseudo-randomness and key-generation are outside the scope of this standard. For further information, see [**KEY-MGMT]**.

- As discussed above, typically one should not use a single cryptographic key for more than a few dozen terabytes of data. We note that this limitation is not unique to this standard. It comes directly from the fact that AES has block size of 128 bits. This limitation is not mitigated by using AES with a 256-bit key.

- As with most cryptographic algorithms, LRW-AES transform should not be used to encrypt its own secret-key. In particular, when using LRW-AES to encrypt its own secret-key followed by a block of zeros, it may be possible to deduce the last 128 bits of the key from the ciphertext.