IEEE P1619™/D11
Draft Standard for Standard Architecture for Encrypted Shared Storage Media

Prepared by the Security in Storage Working Group of the

IEEE Computer Society Committee

Copyright © 2006 by the Institute of Electrical and Electronics Engineers, Inc.
Three Park Avenue
New York, New York 10016-5997, USA
All rights reserved.

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

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

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Abstract: This standard specifies the architecture for protection of data in sector-level storage devices, describing the methods, algorithm, and modes of data protection to be used.

Keywords: Security, Storage, Encryption, Key Management

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Introduction

(This introduction is not part of IEEE P1619/D11, Draft Standard for Standard Architecture for Encrypted Shared Storage Media.)

The purpose of this standard is to describe a method of encryption for data-at-rest in sector-based devices where the threat model includes possible access to stored data by the attacker. 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 XTS-AES tweakable block cipher and its use for encryption of sector-based storage. XTS-AES is a tweakable block cipher that acts on data units of 128 bits or more and uses the AES block cipher as a subroutine. The key material for XTS-AES consists of a data encryption key (used by the AES block cipher) as well as a "tweak key" that is used to incorporate the logical position of the data block into the encryption. XTS-AES is a concrete instantiation of the class of tweakable block ciphers described in reference [XEX04]. The XTS-AES addresses threats such as copy-and-paste and dictionary attacks, while allowing parallelization and pipelining in cipher implementations.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Participants

At the time this draft standard was completed, the Security in Storage Working Group had the following membership:

Jim Hughes, *Chair*

Serge Plotkin, *Vice-chair*

Fabio Maino, *Secretary*

Shai Halevi	Gideon Avida	David Black
Dalit Naor	Laszlo Hars	Bob Snively
Rob Elliott	Hal Finney	Jim Norton
Eric Hibbard	Cyril Guyot	David McGrew
Landon Noll	Glen Jaquette	Russel Dietz
Doug Whiting	Matt Ball	John Geldman
Larry Hofer	Charlie Martin	

This document was edited by Serge Plotkin, Shai Halevi and Dalit Naor.
Test vectors and code supplied by Doug Whiting and Brian Gladman. Robb Elliott helped improve the presentation.

The following members of the balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

(to be supplied by IEEE)

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

CONTENTS

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Draft Standard for Standard Architecture for Encrypted Shared Storage Media

1. Overview

1.1 Scope

This standard specifies the architecture for protection-use-data in sector-level storage devices, describing the methods, algorithm(s), and modes of data protection to be used.

In particular, this standard specifies the XTS-AES transform, its use for encryption of data at rest, and methods for exporting/importing encryption keys for facilitating compatibility and interoperability between different implementations.

1.2 Purpose

This standard provides a standard architecture for media security and enabling components. Present non-standard, insecure encrypted storage methodologies are augmented, and users will be able to create higher-assurance, standard, interoperable solutions.

The XTS-AES transform defined in this standard is intended for encryption of storage where the threat model includes possible access to stored data by the attacker, data to be protected is presented in fixed-size units (sectors, logical disk blocks, etc.), and each unit must be processed separately, independently of other data units. Another requirement in some cases 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 XTS-AES as *transparent encryption*: an encryption/decryption module may be added to an existing system without having to modify the data layout of any of the existing components[1]. 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.

This standard includes the description of the XTS-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-

[1] Adding encryption to a system may necessitate other changes, e.g., adding a key-management layer. Also, encryption may affect the timing characteristics of the system.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

conformant device that encrypts data using XTS-AES can export the key in a way that will allow construction of another standard-conformant device that is able to import this key and decrypt the data.

1.3 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.[2]

3. Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards, Seventh Edition,* should be referenced for terms not defined in this clause.

3.1 key scope: Data encrypted by a particular key, divided into equal-sized data units (see 3.3). The key scope is identified by three non-negative integers: the start of the data, the data unit size, and the length of the data.

3.2 tweak value: The 128-bit value used to represent the logical position of the data being encrypted or decrypted with XTS-AES.

3.3 data unit: 128 or more bits of data within a key scope. The first data unit in a key scope starts with the first bit of the key scope; each subsequent data unit starts with the bit after the end of the previous data unit. Data units within a key scope are of equal sizes. A data unit does not necessarily correspond to a physical block on the storage device.

[2] 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/

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4. Special Terms

4.1 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 as 00011010_2.

4.2 Letter symbols

The following symbols are used in equations and figures:

$\oplus$	Bit-wise exclusive-OR operation
$\otimes$	Modular multiplication of two polynomials over the binary field GF(2), modulo $x^{128}+x^7+x^2+x+1$, where GF stands for Galois Field.
α	An element of GF(2^{128}) that corresponds to polynomial x (i.e., 0000…010), where GF stands for Galois Field.
$\leftarrow$	Assignment of a value to a variable
|	Concatenation (e.g., if K1=001 and K2=101010, then K1|K2=001101010).

5. The XTS-AES transform

5.1 Multiplication by a primitive element α

The encryption and decryption procedures described in the following section use multiplication of a 16-byte value (result of AES encryption or decryption) by j-th power of α, a primitive element of GF(2^{128}). The input value is first converted into an octet array $a_0[k]$, $k = 0,1,...,15$. In particular, the 16-byte result of AES encryption or decryption is treated as a byte array, where $a_0[0]$ is the first byte of the AES block.

This multiplication is defined by the following procedure.

Input:	j is the power of α
	octet array $a_0[k]$, $k = 0,1,...,15$
Output:	octet array $a_j[k]$, $k = 0,1,...,15$

The output array is defined recursively by the following formulas where i is iterated from 0 to j:

$a_{i+1}[0] \leftarrow (2\ (a_i[0] \bmod 128)) \oplus (135 \lfloor a_i[15]/128 \rfloor)$
$a_{i+1}[k] \leftarrow (2\ (a_i[k] \bmod 128)) \oplus \lfloor a_i[k\text{-}1]/128 \rfloor, \quad k = 1,2,\ldots,15$

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Conceptually, the operation is a left shift of each octet by one bit with carry propagating from one octet to the next one. Also, if the 15th (last) octet shift results in a carry, a special value (decimal 135) is xor-ed into the first octet. This value is derived from the modulus of the Galois Field (polynomial $x^{128}+x^7+x^2+x+1$). See Annex C for an alternative way to implement the multiplication by α^j.

5.2 The XTS-AES Encryption Procedure

5.2.1 XTS-AES-blockEnc Procedure, Encryption of a Single 128-bit Block

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

$C \leftarrow$ XTS-AES-blockEnc(Key, P, i, j)

where:

Key is the 256 or 512 bit XTS-AES key.
P is a block of 128 bits (i.e., the plaintext)
i is a 128-bit tweak value, representing the number of the data unit (see 6)
j is the sequential number of the 128-bit block inside the data unit
C is the block of 128 bits of ciphertext resulting from the operation

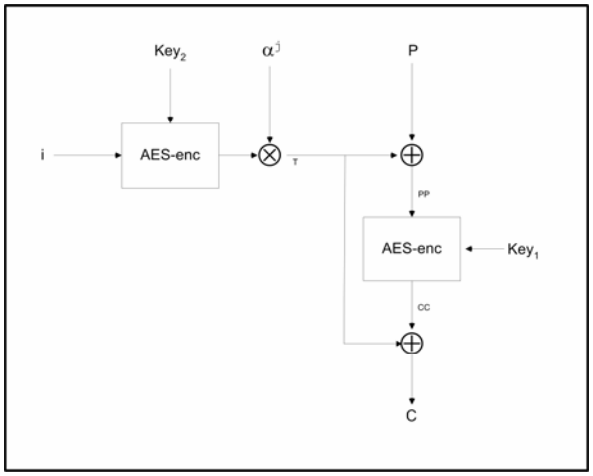
The key is parsed as a concatenation of two fields of equal size called Key_1 and Key_2 such that: $Key = Key_1 \mid Key_2$.

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

1. $T \leftarrow$ AES-enc(Key_2 , i) $\otimes \alpha^j$
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. The multiplication and computation of power in step 1 is executed in GF(2^{128}), where α is a primitive element defined in 4.2 (see 5.1).

Figure 1— Diagram of XTS-AES blockEnc procedure



5.2.2 XTS-AES Encryption of a data unit

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

$C \leftarrow$ XTS-AES-Enc (Key, P, i)

Where:

Key is the 256 or 512 bit XTS-AES key
P is the plaintext
i is a 128-bit tweak, representing the number of the data unit (see 6)
C is the ciphertext resulting from the operation, of the same bit-size as *P*

The plaintext data unit is first partitioned into m+1 blocks:

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

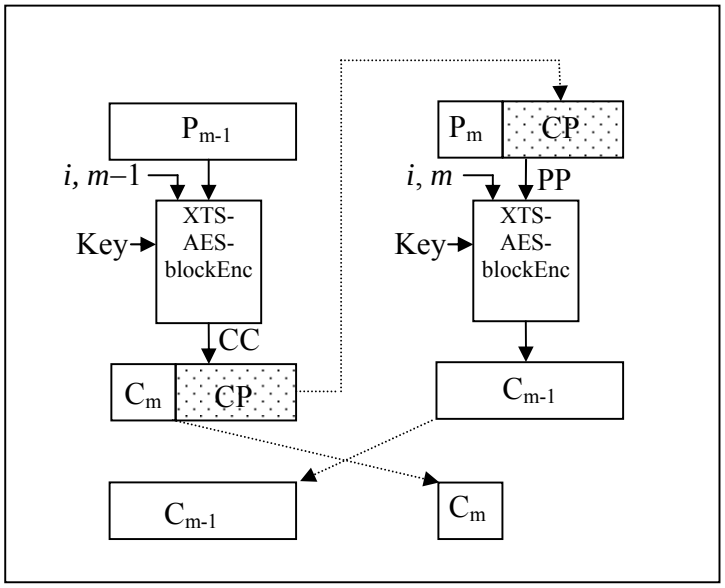
where m is the largest integer such that $128m$ is no more than the bit-size of P, the first m blocks $P_0,\ldots, P_{m-1}$ are all exactly 128 bits long, and the last block P_m is between 0 and 127 bits long. The key is parsed as a concatenation of two fields of equal size called Key_1 and Key_1 such that: $Key = Key_1 \mid Key_2$. The ciphertext C is then computed by the following or an equivalent sequence of steps:

```
1.  for q=0 to m-2
2.      Cj ← XTS-AES-blockEnc(Key, Pj, i, q)
3.  endfor
4.  b ← bit-size of Pm
5.  if b=0
6.      Cm-1 ← XTS-AES-blockEnc(Key, Pm-1, i, m-1)
7.      Cm ← empty
8.  else
          // Pm is a partial block
9.      CC ← XTS-AES-blockEnc(Key, Pm-1, i, m-1)
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 ← XTS-AES-blockEnc(Key, PP, i, m)
14. endif
15. C ← C1|… |Cm-1|Cm
```

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.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Figure 2—XTS-AES encryption of last two blocks when last block is 1 to 127 bits



5.3 The XTS-AES Decryption Procedure

5.3.1 XTS-AES-blockDec Procedure, Decryption of a Single 128-bit Block

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

$P \leftarrow$ XTS-AES-blockDec(*Key*, C, *i*)

where:

Key is the 256 or 512-bit XTS-AES key
C the 128-bit block of ciphertext
i is a 128-bit tweak value, representing the number of the data unit (see 6)
j is the sequential number of the 128-bit block inside the data unit
P is the 128-bit block of plaintext resulting from the operation

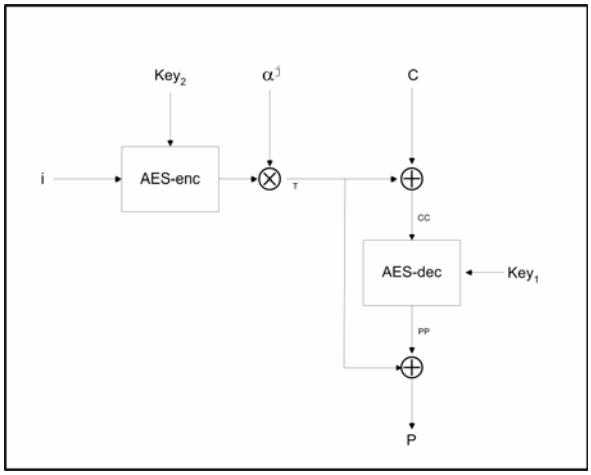
The key is parsed as a concatenation of two fields of equal size called Key_1 and Key_2 such that: $Key = Key_1 \mid Key_2$. The plaintext shall then be computed by the following or an equivalent sequence of steps:

1. $T \leftarrow$ AES-enc(Key_2 , i) $\otimes \alpha^j$
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. The multiplication and computation of power in step 1 is executed in GF(2^{128}), where α is a primitive element defined in 4.2 (see 5.1).

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Figure 3— Diagram of XTS-AES blockDec procedure



5.3.2 XTS-AES Decryption of a Data Unit

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

$C \leftarrow$ XTS-AES-Dec(*Key*, *C*, *i*)

where:

Key is the 256 or 512-bit XTS-AES key
C is the ciphertext corresponding to the data unit
i is a 128-bit tweak, representing the number of the data unit (see 6)
P is the plaintext data unit resulting from the operation, of the same bit-size as *C*

The ciphertext is first partitioned into m+1 blocks:

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

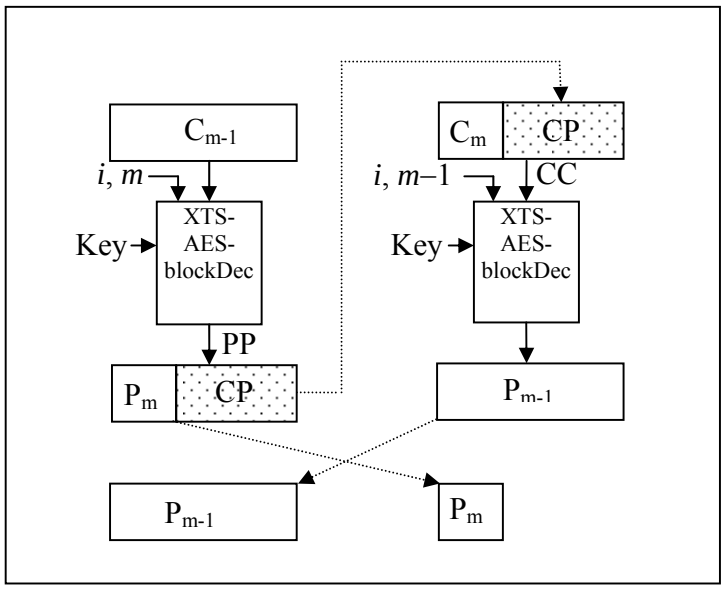
where m is the largest integer such that 128m is no more than the bit-size of C, the first m blocks $C_0,\ldots,$ C_{m-1} are all exactly 128 bits long, and the last block C_m is between 0 and 127 bits long. The key is parsed as a concatenation of two fields of equal size called Key_1 and Key_1 such that: $Key = Key_1 \mid Key_2$. The plaintext P is then computed by the following or an equivalent sequence of steps:

```
1.  for q=0 to m-2
2.      Pj ← XTS-AES-blockDec(Key, Cj, i, q)
3.  endfor
4.  b ← bit-size of Cm
5.  if b=0
6.      Pm-1 ← XTS-AES-blockDec(Key, Cm-1, i, m-1)
7.      Pm ← empty
8.  else
          // Cm is a partial block
9.      PP ← XTS-AES-blockDec(Key, Cm-1, i, m)
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 ← XTS-AES-blockDec(Key, CC, i, m-1)
14. endif
15. P ← P1 |… |Pm-1|Pm
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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—XTS-AES decryption of last two blocks when last block is 1 to 127 bits



6. Using XTS-AES-128 and XTS-AES-256 for Encryption of Storage

The encryption and decryption procedures described in 5.2.2 and 5.3.2 use AES as the basic building block. If the XTS-AES key consists of 256 bits, the procedures use 128-bit AES; if XTS-AES key consists of 512 bits, the procedures use 256-bit AES. For completeness, the mode shall be referred to as XTS-AES-128 and the second as XTS-AES-256. To be compliant with the standard, the implementation shall support at least one of the above modes.

The standard defines using the XTS-AES-128 and XTS-AES-256 transforms to encrypt or decrypt data at rest, where the data consists of an integral number of data units, numbered consecutively, all of which are of the same bit length. The number of bits in a data unit is not necessarily divisible by 128 (AES block size). The number of 16-byte blocks in the data unit shall not exceed 2^{128}-2. [3]The mapping between the data unit and the transfer of data, placement and composition of data on the storage device is beyond the scope of the standard. Devices compliant with this standard should include documentation describing this mapping.

To use this standard, a 256-bit or 512-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, the sequence number of this data unit within the scope of the key must be known. It is possible to encrypt/decrypt part of a data unit, given the key, the sequential number of the data unit, and the position of the bits to be encrypted/decrypted inside the data unit.

[3] This is not a practical limitation and is stated here for completeness only.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

As defined in 7.1.4, 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. 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 XTS-AES-[128,256] 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.

7. Exporting and Archiving XTS-AES-128 and XTS-AES-256 keys

7.1 Key Backup Structure

7.1.1 Key Backup Structure Overview

The system surrounding a device compliant with this standard should support the Key Backup structure defined in this clause. 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 XTS-AES-[128,256]. These data units are assumed to be a contiguous sequence, starting at a given offset from the first data unit of the input[4]. For example, if the input data units are on a single physical medium, the offset specifies the first data unit on this medium that is encrypted with the given XTS-AES-[128,256] key. 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.

Table 1 lists the elements of the key backup structure.

Table 1 Key Backup Structure

Field	Format	Description
StructureID	Structure ID	Identifier of current structure
Standard	Standard Description	Standard identifier
KeyScope	Key Scope	Key Scope
Transform	Transform	Transform description
KeyMaterial	Key Material	Key material and its length

[4] The inclusion of offset facilitates use of several different keys for a single physical medium. For example, each key can correspond to a different sequence of LBAs residing on a physical disk drive. Another situation where offset is needed is when only part of encrypted data (contiguous sequence of data units starting at some offset) is archived together with the corresponding Key Backup structure. It is impossible to decrypt this data without knowledge of the offset.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7.1.2 Structure ID

Table 2 defines the StructureID element, which is the information needed to uniquely identify a particular instance of a key backup structure.

Table 2 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.

7.1.3 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 3 defines the Standard element.

Table 3 Standard Description

Field	Size	XML Encoding	Description
StandardNumber	Up to 128 bytes	Text	Number of the Standard used. Should be IEEE STD 1619-2007.
StandardComment	Up to 256 bytes	Text	Any additional standard-related information.

7.1.4 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 4 defines the KeyScope element.

Table 4 Key Scope

Field	Size	XML Encoding	Description
KeyScopeStart	16 bytes	Integer	The offset in bits from the start of the input sequence of data units.
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.

7.1.5 Transform

Table 5 defines the Transform element.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 5 Transform

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 6 below.

Table 6 Supported Transforms

String	Description
XTS-AES-128	The XTS-AES-128 transform as defined in this standard.
XTS-AES-256	The XTS-AES-256 transform as defined in this standard.

7.1.6 Key Material

Table 7 defines the KeyMaterial element. All key material is secret.

Table 7 Key Material

Field	Size	XML Encoding	Description
KeyLength	2 bytes	Integer	Length (in bits) of the key. Allowed values are 256 (for XTS-AES-128) and 512 (for XTS-AES-256).
KeyValue	variable	Base64	The value of the key.

7.2 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 Key Backup format:

```
<!ELEMENT KeyBackup (StructureID, Standard, KeyScope, Transform, KeyMaterial)>
   <!ELEMENT StructureID  (ID, Comment?)>
   <!ELEMENT ID (#PCDATA)>
      <!ATTLIST ID Encoding CDATA #FIXED "Base64">
   <!ELEMENT Comment      (#PCDATA)>
   <!ELEMENT Standard     (StandardNumber, StandardComment?)>
   <!ELEMENT StandardNumber (#PCDATA)>
   <!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)>
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
      <!ATTLIST KeyLength Encoding CDATA #FIXED "Integer">
   <!ELEMENT KeyValue         (#PCDATA)>
      <!ATTLIST KeyValue Encoding CDATA #FIXED "Base64">
```

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">YUBlJHJqMDNhWjFAJCVwXQ==</ID>
   <Comment>Comment text here</Comment>
   </StructureID>
<Standard>
   <StandardNumber>IEEE STD 1619-2007</StandardNumber>
   <StandardComment>Disk</StandardComment>
   </Standard>
<KeyScope>
   <KeyScopeStart Encoding="Integer">0</KeyScopeStart>
   <DataUnitSize Encoding="Integer">4096</DataUnitSize>
   <KeyScopeLength Encoding="Integer">1083</KeyScopeLength>
   </KeyScope>
<Transform>
   <TransformName>XTS-AES-256</TransformName>
   </Transform>
<KeyMaterial>
   <KeyLength Encoding="Integer">512</KeyLength>
   <KeyValue Encoding="Base64">
            IUApKFQlWEpHJCkoVypUJVgoKU5UJVdYK
            ShXJVhOSlJFR0gpSCgjJWd0eDk3d3h0NW
            03NTNobXR4ISNkZjRzZw==
      </KeyValue>
</KeyMaterial>
</KeyBackup>
```

7.3 Encryption of Key Backup material

A Key backup structure may be protected as follows. The actual key material (KeyMaterial from Table 7) 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.

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]).

The 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#"
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
    <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] uses 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 is stored as an EncryptedKey element elsewhere in the document with ID=’WK’.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex A

(informative)

Bibliography

[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.

[XEX04] P. Rogaway. "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC". Advances in Cryptology - Asiacrypt 2004. Lecture Notes in Computer Science, Springer-Verlag, 2004

[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 Guidelines SP800-57.
http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf and
http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part2.pdf

[XML-ENC] XML Encryption Syntax and Processing. W3C.
http://www.w3.org/TR/xmlenc-core

[XML-KMS] XML Key Management Specification (XKMS). W3C.
http://www.w3.org/TR/xkms

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex B

(informative)

Test Vectors

This Annex lists several examples of applying XTS-AES-128 and XTS-AES-256. All numbers in this Annex are hexadecimal. For readability, the examples explicitly parse the XTS-AES key into Key_1 and Key_2. All outputs are little endian byte arrays. PTX prefix denotes plaintext; CTX prefix denotes ciphertext. Some of the vectors include intermediate calculation TWK, which is the mask computed in step 1 in 5.2.1.

XTS-AES applied for a data unit of 32 bytes, 32 bytes key material.

```
Vector 1
Key1 00000000000000000000000000000000
Key2 00000000000000000000000000000000
Data Unit Sequence number 0
PTX 0000000000000000000000000000000000000000000000000000000000000000
TWK 66e94bd4ef8a2c3b884cfa59ca342b2eccd297a8df1559761099f4b39469565c
CTX 917cf69ebd68b2ec9b9fe9a3eadda692cd43d2f59598ed858c02c2652fbf922e

Vector 2
Key1 11111111111111111111111111111111
Key2 22222222222222222222222222222222
Data Unit Sequence number 3333333333
PTX 4444444444444444444444444444444444444444444444444444444444444444
TWK 3f803bcd0d7fd2b37558419f59d5cda6f900779a1bfea467ebb0823eb3aa9b4d
CTX c454185e6a16936e39334038acef838bfb186fff7480adc4289382ecd6d394f0

Vector 3
Key1 fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0
Key2 22222222222222222222222222222222
Data Unit Sequence number 3333333333
PTX 4444444444444444444444444444444444444444444444444444444444444444
TWK 3f803bcd0d7fd2b37558419f59d5cda6f900779a1bfea467ebb0823eb3aa9b4d
CTX af85336b597afc1a900b2eb21ec949d292df4c047e0b21532186a5971a227a89
```

XTS-AES-128 applied for a data unit of 512 bytes

```
Vector 4
Key1 27182818284590452353602874713526
Key2 31415926535897932384626433832795
Data Unit Sequence number 0

PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 27a7479befa1d476489f308cd4cfa6e2a96e4bbe3208ff25287dd3819616e89c
CTX c78cf7f5e543445f8333d8fa7f56000005279fa5d8b5e4ad40e736ddb4d35412
CTX 328063fd2aab53e5ea1e0a9f332500a5df9487d07a5c92cc512c8866c7e860ce
CTX 93fdf166a24912b422976146ae20ce846bb7dc9ba94a767aaef20c0d61ad0265
CTX 5ea92dc4c4e41a8952c651d33174be51a10c421110e6d81588ede82103a252d8
CTX a750e8768defffed9122810aaeb99f9172af82b604dc4b8e51bcb08235a6f434
CTX 1332e4ca60482a4ba1a03b3e65008fc5da76b70bf1690db4eae29c5f1badd03c
CTX 5ccf2a55d705ddcd86d449511ceb7ec30bf12b1fa35b913f9f747a8afd1b130e
CTX 94bff94effd01a91735ca1726acd0b197c4e5b03393697e126826fb6bbde8ecc
CTX 1e08298516e2c9ed03ff3c1b7860f6de76d4cecd94c8119855ef5297ca67e9f3
CTX e7ff72b1e99785ca0a7e7720c5b36dc6d72cac9574c8cbbc2f801e23e56fd344
CTX b07f22154beba0f08ce8891e643ed995c94d9a69c9f1b5f499027a78572aeebd
CTX 74d20cc39881c213ee770b1010e4bea718846977ae119f7a023ab58cca0ad752
CTX afe656bb3c17256a9f6e9bf19fdd5a38fc82bbe872c5539edb609ef4f79c203e
CTX bb140f2e583cb2ad15b4aa5b655016a8449277dbd477ef2c8d6c017db738b18d
CTX eb4a427d1923ce3ff262735779a418f20a282df920147beabe421ee5319d0568

Vector 5
Key1 27182818284590452353602874713526
Key2 31415926535897932384626433832795
Data Unit Sequence Number 01

PTX 27a7479befa1d476489f308cd4cfa6e2a96e4bbe3208ff25287dd3819616e89c
PTX c78cf7f5e543445f8333d8fa7f56000005279fa5d8b5e4ad40e736ddb4d35412
PTX 328063fd2aab53e5ea1e0a9f332500a5df9487d07a5c92cc512c8866c7e860ce
PTX 93fdf166a24912b422976146ae20ce846bb7dc9ba94a767aaef20c0d61ad0265
PTX 5ea92dc4c4e41a8952c651d33174be51a10c421110e6d81588ede82103a252d8
PTX a750e8768defffed9122810aaeb99f9172af82b604dc4b8e51bcb08235a6f434
PTX 1332e4ca60482a4ba1a03b3e65008fc5da76b70bf1690db4eae29c5f1badd03c
PTX 5ccf2a55d705ddcd86d449511ceb7ec30bf12b1fa35b913f9f747a8afd1b130e
PTX 94bff94effd01a91735ca1726acd0b197c4e5b03393697e126826fb6bbde8ecc
PTX 1e08298516e2c9ed03ff3c1b7860f6de76d4cecd94c8119855ef5297ca67e9f3
PTX e7ff72b1e99785ca0a7e7720c5b36dc6d72cac9574c8cbbc2f801e23e56fd344
PTX b07f22154beba0f08ce8891e643ed995c94d9a69c9f1b5f499027a78572aeebd
PTX 74d20cc39881c213ee770b1010e4bea718846977ae119f7a023ab58cca0ad752
PTX afe656bb3c17256a9f6e9bf19fdd5a38fc82bbe872c5539edb609ef4f79c203e
PTX bb140f2e583cb2ad15b4aa5b655016a8449277dbd477ef2c8d6c017db738b18d
PTX eb4a427d1923ce3ff262735779a418f20a282df920147beabe421ee5319d0568
CTX 264d3ca8512194fec312c8c9891f279fefdd608d0c027b60483a3fa811d65ee5
CTX 9d52d9e40ec5672d81532b38b6b089ce951f0f9c35590b8b978d175213f329bb
CTX 1c2fd30f2f7f30492a61a532a79f51d36f5e31a7c9a12c286082ff7d2394d18f
CTX 783e1a8e72c722caaaa52d8f065657d2631fd25bfd8e5baad6e527d763517501
CTX c68c5edc3cdd55435c532d7125c8614deed9adaa3acade5888b87bef641c4c99
CTX 4c8091b5bcd387f3963fb5bc37aa922fbfe3df4e5b915e6eb514717bdd2a7407
CTX 9a5073f5c4bfd46adf7d282e7a393a52579d11a028da4d9cd9c77124f9648ee3
CTX 83b1ac763930e7162a8d37f350b2f74b8472cf09902063c6b32e8c2d9290cefb
CTX d7346d1c779a0df50edcde4531da07b099c638e83a755944df2aef1aa31752fd
CTX 323dcb710fb4bfbb9d22b925bc3577e1b8949e729a90bbafeacf7f7879e7b114
CTX 7e28ba0bae940db795a61b15ecf4df8db07b824bb062802cc98a9545bb2aaeed
CTX 77cb3fc6db15dcd7d80d7d5bc406c4970a3478ada8899b329198eb61c193fb62
CTX 75aa8ca340344a75a862aebe92eee1ce032fd950b47d7704a3876923b4ad6284
CTX 4bf4a09c4dbe8b4397184b7471360c9564880aedddb9baa4af2e75394b08cd32
CTX ff479c57a07d3eab5d54de5f9738b8d27f27a9f0ab11799d7b7ffefb2704c95c
CTX 6ad12c39f1e867a4b7b1d7818a4b753dfd2a89ccb45e001a03a867b187f225dd

Vector 6
Key1 27182818284590452353602874713526
Key2 31415926535897932384626433832795
Data Unit Sequence Number 02

PTX 264d3ca8512194fec312c8c9891f279fefdd608d0c027b60483a3fa811d65ee5
PTX 9d52d9e40ec5672d81532b38b6b089ce951f0f9c35590b8b978d175213f329bb
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
PTX 1c2fd30f2f7f30492a61a532a79f51d36f5e31a7c9a12c286082ff7d2394d18f
PTX 783e1a8e72c722caaaa52d8f065657d2631fd25bfd8e5baad6e527d763517501
PTX c68c5edc3cdd55435c532d7125c8614deed9adaa3acade5888b87bef641c4c99
PTX 4c8091b5bcd387f3963fb5bc37aa922fbfe3df4e5b915e6eb514717bdd2a7407
PTX 9a5073f5c4bfd46adf7d282e7a393a52579d11a028da4d9cd9c77124f9648ee3
PTX 83b1ac763930e7162a8d37f350b2f74b8472cf09902063c6b32e8c2d9290cefb
PTX d7346d1c779a0df50edcde4531da07b099c638e83a755944df2aef1aa31752fd
PTX 323dcb710fb4bfbb9d22b925bc3577e1b8949e729a90bbafeacf7f7879e7b114
PTX 7e28ba0bae940db795a61b15ecf4df8db07b824bb062802cc98a9545bb2aaeed
PTX 77cb3fc6db15dcd7d80d7d5bc406c4970a3478ada8899b329198eb61c193fb62
PTX 75aa8ca340344a75a862aebe92eee1ce032fd950b47d7704a3876923b4ad6284
PTX 4bf4a09c4dbe8b4397184b7471360c9564880aedddb9baa4af2e75394b08cd32
PTX ff479c57a07d3eab5d54de5f9738b8d27f27a9f0ab11799d7b7ffefb2704c95c
PTX 6ad12c39f1e867a4b7b1d7818a4b753dfd2a89ccb45e001a03a867b187f225dd
CTX fa762a3680b76007928ed4a4f49a9456031b704782e65e16cecb54ed7d017b5e
CTX 18abd67b338e81078f21edb7868d901ebe9c731a7c18b5e6dec1d6a72e078ac9
CTX a4262f860beefa14f4e821018272e411a951502b6e79066e84252c3346f3aa62
CTX 344351a291d4bedc7a07618bdea2af63145cc7a4b8d4070691ae890cd65733e7
CTX 946e9021a1dffc4c59f159425ee6d50ca9b135fa6162cea18a939838dc000fb3
CTX 86fad086acce5ac07cb2ece7fd580b00cfa5e98589631dc25e8e2a3daf2ffdec
CTX 26531659912c9d8f7a15e5865ea8fb5816d6207052bd7128cd743c12c8118791
CTX a4736811935eb982a532349e31dd401e0b660a568cb1a4711f552f55ded59f1f
CTX 15bf7196b3ca12a91e488ef59d64f3a02bf45239499ac6176ae321c4a211ec54
CTX 5365971c5d3f4f09d4eb139bfdf2073d33180b21002b65cc9865e76cb24cd92c
CTX 874c24c18350399a936ab3637079295d76c417776b94efce3a0ef7206b151105
CTX 19655c956cbd8b2489405ee2b09a6b6eebe0c53790a12a8998378b33a5b71159
CTX 625f4ba49d2a2fdba59fbf0897bc7aabd8d707dc140a80f0f309f835d3da54ab
CTX 584e501dfa0ee977fec543f74186a802b9a37adb3e8291eca04d66520d229e60
CTX 401e7282bef486ae059aa70696e0e305d777140a7a883ecdcb69b9ff938e8a42
CTX 31864c69ca2c2043bed007ff3e605e014bcf518138dc3a25c5e236171a2d01d6

Vector 7
Key1 27182818284590452353602874713526
Key2 31415926535897932384626433832795
Data Unit Sequence Number fd

PTX 8e41b78c390b5af9d758bb214a67e9f6bf7727b09ac6124084c37611398fa45d
PTX aad94868600ed391fb1acd4857a95b466e62ef9f4b377244d1c152e7b30d731a
PTX ad30c716d214b707aed99eb5b5e580b3e887cf7497465651d4b60e6042051da3
PTX 693c3b78c14489543be8b6ad0ba629565bba202313ba7b0d0c94a3252b676f46
PTX cc02ce0f8a7d34c0ed229129673c1f61aed579d08a9203a25aac3a77e9db6026
PTX 7996db38df637356d9dcd1632e369939f2a29d89345c66e05066f1a3677aef18
PTX dea4113faeb629e46721a66d0a7e785d3e29af2594eb67dfa982affe0aac058f
PTX 6e15864269b135418261fc3afb089472cf68c45dd7f231c6249ba0255e1e0338
PTX 33fc4d00a3fe02132d7bc3873614b8aee34273581ea0325c81f0270affa13641
PTX d052d36f0757d484014354d02d6883ca15c24d8c3956b1bd027bcf41f151fd80
PTX 23c5340e5606f37e90fdb87c86fb4fa634b3718a30bace06a66eaf8f63c4aa3b
PTX 637826a87fe8cfa44282e92cb1615af3a28e53bc74c7cba1a0977be9065d0c1a
PTX 5dec6c54ae38d37f37aa35283e048e5530a85c4e7a29d7b92ec0c3169cdf2a80
PTX 5c7604bce60049b9fb7b8eaac10f51ae23794ceba68bb58112e293b9b692ca72
PTX 1b37c662f8574ed4dba6f88e170881c82cddc1034a0ca7e284bf0962b6b26292
PTX d836fa9f73c1ac770eef0f2d3a1eaf61d3e03555fd424eedd67e18a18094f888
CTX d55f684f81f4426e9fde92a5ff02df2ac896af63962888a97910c1379e20b0a3
CTX b1db613fb7fe2e07004329ea5c22bfd33e3dbe4cf58cc608c2c26c19a2e2fe22
CTX f98732c2b5cb844cc6c0702d91e1d50fc4382a7eba5635cd602432a2306ac4ce
CTX 82f8d70c8d9bc15f918fe71e74c622d5cf71178bf6e0b9cc9f2b41dd8dbe441c
CTX 41cd0c73a6dc47a348f6702f9d0e9b1b1431e948e299b9ec2272ab2c5f0c7be8
CTX 6affa5dec87a0bee81d3d50007edaa2bcfccb35605155ff36ed8edd4a40dcd4b
CTX 243acd11b2b987bdbfaf91a7cac27e9c5aea525ee53de7b2d3332c8644402b82
CTX 3e94a7db26276d2d23aa07180f76b4fd29b9c0823099c9d62c519880aee7e969
CTX 7617c1497d47bf3e571950311421b6b734d38b0db91eb85331b91ea9f61530f5
CTX 4512a5a52a4bad589eb69781d537f23297bb459bdad2948a29e1550bf4787e0b
CTX e95bb173cf5fab17dab7a13a052a63453d97ccec1a321954886b7a1299faaeec
CTX ae35c6eaaca753b041b5e5f093bf83397fd21dd6b3012066fcc058cc32c3b09d
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
CTX 7562dee29509b5839392c9ff05f51f3166aaac4ac5f238038a3045e6f72e48ef
CTX 0fe8bc675e82c318a268e43970271bf119b81bf6a982746554f84e72b9f00280
CTX a320a08142923c23c883423ff949827f29bbacdc1ccdb04938ce6098c95ba6b3
CTX 2528f4ef78eed778b2e122ddfd1cbdd11d1c0a6783e011fc536d63d053260637

Vector 8
Key1 27182818284590452353602874713526
Key2 31415926535897932384626433832795
Data Unit Sequence Number fe

PTX d55f684f81f4426e9fde92a5ff02df2ac896af63962888a97910c1379e20b0a3
PTX b1db613fb7fe2e07004329ea5c22bfd33e3dbe4cf58cc608c2c26c19a2e2fe22
PTX f98732c2b5cb844cc6c0702d91e1d50fc4382a7eba5635cd602432a2306ac4ce
PTX 82f8d70c8d9bc15f918fe71e74c622d5cf71178bf6e0b9cc9f2b41dd8dbe441c
PTX 41cd0c73a6dc47a348f6702f9d0e9b1b1431e948e299b9ec2272ab2c5f0c7be8
PTX 6affa5dec87a0bee81d3d50007edaa2bcfccb35605155ff36ed8edd4a40dcd4b
PTX 243acd11b2b987bdbfaf91a7cac27e9c5aea525ee53de7b2d3332c8644402b82
PTX 3e94a7db26276d2d23aa07180f76b4fd29b9c0823099c9d62c519880aee7e969
PTX 7617c1497d47bf3e571950311421b6b734d38b0db91eb85331b91ea9f61530f5
PTX 4512a5a52a4bad589eb69781d537f23297bb459bdad2948a29e1550bf4787e0b
PTX e95bb173cf5fab17dab7a13a052a63453d97ccec1a321954886b7a1299faaeec
PTX ae35c6eaaca753b041b5e5f093bf83397fd21dd6b3012066fcc058cc32c3b09d
PTX 7562dee29509b5839392c9ff05f51f3166aaac4ac5f238038a3045e6f72e48ef
PTX 0fe8bc675e82c318a268e43970271bf119b81bf6a982746554f84e72b9f00280
PTX a320a08142923c23c883423ff949827f29bbacdc1ccdb04938ce6098c95ba6b3
PTX 2528f4ef78eed778b2e122ddfd1cbdd11d1c0a6783e011fc536d63d053260637
CTX 72efc1ebfe1ee25975a6eb3aa8589dda2b261f1c85bdab442a9e5b2dd1d7c395
CTX 7a16fc08e526d4b1223f1b1232a11af274c3d70dac57f83e0983c498f1a6f1ae
CTX cb021c3e70085a1e527f1ce41ee5911a82020161529cd82773762daf5459de94
CTX a0a82adae7e1703c808543c29ed6fb32d9e004327c1355180c995a07741493a0
CTX 9c21ba01a387882da4f62534b87bb15d60d197201c0fd3bf30c1500a3ecfecdd
CTX 66d8721f90bcc4c17ee925c61b0a03727a9c0d5f5ca462fbfa0af1c2513a9d9d
CTX 4b5345bd27a5f6e653f751693e6b6a2b8ead57d511e00e58c45b7b8d005af792
CTX 88f5c7c22fd4f1bf7a898b03a5634c6a1ae3f9fae5de4f296a2896b23e7ed43e
CTX d14fa5a2803f4d28f0d3ffcf24757677aebdb47bb388378708948a8d4126ed18
CTX 39e0da29a537a8c198b3c66ab00712dd261674bf45a73d67f76914f830ca014b
CTX 65596f27e4cf62de66125a5566df9975155628b400fbfb3a29040ed50faffdbb
CTX 18aece7c5c44693260aab386c0a37b11b114f1c415aebb653be468179428d43a
CTX 4d8bc3ec38813eca30a13cf1bb18d524f1992d44d8b1a42ea30b22e6c95b199d
CTX 8d182f8840b09d059585c31ad691fa0619ff038aca2c39a943421157361717c4
CTX 9d322028a74648113bd8c9d7ec77cf3c89c1ec8718ceff8516d96b34c3c614f1
CTX 0699c9abc4ed0411506223bea16af35c883accdbe1104eef0cfdb54e12fb230a

Vector 9
Key1 27182818284590452353602874713526
Key2 31415926535897932384626433832795
Data Unit Sequence Number ff

PTX 72efc1ebfe1ee25975a6eb3aa8589dda2b261f1c85bdab442a9e5b2dd1d7c395
PTX 7a16fc08e526d4b1223f1b1232a11af274c3d70dac57f83e0983c498f1a6f1ae
PTX cb021c3e70085a1e527f1ce41ee5911a82020161529cd82773762daf5459de94
PTX a0a82adae7e1703c808543c29ed6fb32d9e004327c1355180c995a07741493a0
PTX 9c21ba01a387882da4f62534b87bb15d60d197201c0fd3bf30c1500a3ecfecdd
PTX 66d8721f90bcc4c17ee925c61b0a03727a9c0d5f5ca462fbfa0af1c2513a9d9d
PTX 4b5345bd27a5f6e653f751693e6b6a2b8ead57d511e00e58c45b7b8d005af792
PTX 88f5c7c22fd4f1bf7a898b03a5634c6a1ae3f9fae5de4f296a2896b23e7ed43e
PTX d14fa5a2803f4d28f0d3ffcf24757677aebdb47bb388378708948a8d4126ed18
PTX 39e0da29a537a8c198b3c66ab00712dd261674bf45a73d67f76914f830ca014b
PTX 65596f27e4cf62de66125a5566df9975155628b400fbfb3a29040ed50faffdbb
PTX 18aece7c5c44693260aab386c0a37b11b114f1c415aebb653be468179428d43a
PTX 4d8bc3ec38813eca30a13cf1bb18d524f1992d44d8b1a42ea30b22e6c95b199d
PTX 8d182f8840b09d059585c31ad691fa0619ff038aca2c39a943421157361717c4
PTX 9d322028a74648113bd8c9d7ec77cf3c89c1ec8718ceff8516d96b34c3c614f1
PTX 0699c9abc4ed0411506223bea16af35c883accdbe1104eef0cfdb54e12fb230a
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
CTX 3260ae8dad1f4a32c5cafe3ab0eb95549d461a67ceb9e5aa2d3afb62dece0553
CTX 193ba50c75be251e08d1d08f1088576c7efdfaaf3f459559571e12511753b07a
CTX f073f35da06af0ce0bbf6b8f5ccc5cea500ec1b211bd51f63b606bf6528796ca
CTX 12173ba39b8935ee44ccce646f90a45bf9ccc567f0ace13dc2d53ebeedc81f58
CTX b2e41179dddf0d5a5c42f5d8506c1a5d2f8f59f3ea873cbcd0eec19acbf32542
CTX 3bd3dcb8c2b1bf1d1eaed0eba7f0698e4314fbeb2f1566d1b9253008cbccf45a
CTX 2b0d9c5c9c21474f4076e02be26050b99dee4fd68a4cf890e496e4fcae7b70f9
CTX 4ea5a9062da0daeba1993d2ccd1dd3c244b8428801495a58b216547e7e847c46
CTX d1d756377b6242d2e5fb83bf752b54e0df71e889f3a2bb0f4c10805bf3c59037
CTX 6e3c24e22ff57f7fa965577375325cea5d920db94b9c336b455f6e894c01866f
CTX e9fbb8c8d3f70a2957285f6dfb5dcd8cbf54782f8fe7766d4723819913ac7734
CTX 21e3a31095866bad22c86a6036b2518b2059b4229d18c8c2ccbdf906c6cc6e82
CTX 464ee57bddb0bebcb1dc645325bfb3e665ef7251082c88ebb1cf203bd779fdd3
CTX 8675713c8daadd17e1cabee432b09787b6ddf3304e38b731b45df5df51b78fcf
CTX b3d32466028d0ba36555e7e11ab0ee0666061d1645d962444bc47a38188930a8
CTX 4b4d561395c73c087021927ca638b7afc8a8679ccb84c26555440ec7f10445cd
```

XTS-AES-256 applied for a data unit of 512 bytes

```
Vector 10
Key1 271828182845904523536028747135266249775724709369995957496696762 7
Key2 314159265358979323846264338327950288419716939937510582097494459 2
Data Unit Sequence Number ff

PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 1c3b3a102f770386e4836c99e370cf9bea00803f5e482357a4ae12d414a3e63b
CTX 5d31e276f8fe4a8d66b317f9ac683f44680a86ac35adfc3345befecb4bb188fd
CTX 5776926c49a3095eb108fd1098baec70aaa66999a72a82f27d848b21d4a741b0
CTX c5cd4d5fff9dac89aeba122961d03a757123e9870f8acf1000020887891429ca
CTX 2a3e7a7d7df7b10355165c8b9a6d0a7de8b062c4500dc4cd120c0f7418dae3d0
CTX b5781c34803fa75421c790dfe1de1834f280d7667b327f6c8cd7557e12ac3a0f
CTX 93ec05c52e0493ef31a12d3d9260f79a289d6a379bc70c50841473d1a8cc81ec
CTX 583e9645e07b8d9670655ba5bbcfecc6dc3966380ad8fecb17b6ba02469a020a
CTX 84e18e8f84252070c13e9f1f289be54fbc481457778f616015e1327a02b140f1
CTX 505eb309326d68378f8374595c849d84f4c333ec4423885143cb47bd71c5edae
CTX 9be69a2ffeceb1bec9de244fbe15992b11b77c040f12bd8f6a975a44a0f90c29
CTX a9abc3d4d893927284c58754cce294529f8614dcd2aba991925fedc4ae74ffac
CTX 6e333b93eb4aff0479da9a410e4450e0dd7ae4c6e2910900575da401fc07059f
CTX 645e8b7e9bfdef33943054ff84011493c27b3429eaedb4ed5376441a77ed4385
CTX 1ad77f16f541dfd269d50d6a5f14fb0aab1cbb4c1550be97f7ab4066193c4caa
CTX 773dad38014bd2092fa755c824bb5e54c4f36ffda9fcea70b9c6e693e148c151
GEN

Vector 11
Key1 271828182845904523536028747135266249775724709369995957496696762 7
Key2 314159265358979323846264338327950288419716939937510582097494459 2
Data Unit Sequence Number ffff
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 77a31251618a15e6b92d1d66dffe7b50b50bad552305ba0217a610688eff7e11
CTX e1d0225438e093242d6db274fde801d4cae06f2092c728b2478559df58e837c2
CTX 469ee4a4fa794e4bbc7f39bc026e3cb72c33b0888f25b4acf56a2a9804f1ce6d
CTX 3d6e1dc6ca181d4b546179d55544aa7760c40d06741539c7e3cd9d2f6650b201
CTX 3fd0eeb8c2b8e3d8d240ccae2d4c98320a7442e1c8d75a42d6e6cfa4c2eca179
CTX 8d158c7aecdf82490f24bb9b38e108bcda12c3faf9a21141c3613b58367f922a
CTX aa26cd22f23d708dae699ad7cb40a8ad0b6e2784973dcb605684c08b8d6998c6
CTX 9aac049921871ebb65301a4619ca80ecb485a31d744223ce8ddc2394828d6a80
CTX 470c092f5ba413c3378fa6054255c6f9df4495862bbb3287681f931b687c888a
CTX bf844dfc8fc28331e579928cd12bd2390ae123cf03818d14dedde5c0c24c8ab0
CTX 18bfca75ca096f2d531f3d1619e785f1ada437cab92e980558b3dce1474afb75
CTX bfedbf8ff54cb2618e0244c9ac0d3c66fb51598cd2db11f9be39791abe447c63
CTX 094f7c453b7ff87cb5bb36b7c79efb0872d17058b83b15ab0866ad8a58656c5a
CTX 7e20dbdf308b2461d97c0ec0024a2715055249cf3b478ddd4740de654f75ca68
CTX 6e0d7345c69ed50cdc2a8b332b1f8824108ac937eb050585608ee734097fc090
CTX 54fbff89eeaeea791f4a7ab1f9868294a4f9e27b42af8100cb9d59cef9645803

Vector 12
Key1 2718281828459045235360287471352662497757247093699959574966967627
Key2 3141592653589793238462643383279502884197169399375105820974944592
Data Unit Sequence Number ffffff

PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX e387aaa58ba483afa7e8eb469778317ecf4cf573aa9d4eac23f2cdf914e4e200
CTX a8b490e42ee646802dc6ee2b471b278195d60918ececb44bf79966f83faba049
CTX 9298ebc699c0c8634715a320bb4f075d622e74c8c932004f25b41e361025b5a8
CTX 7815391f6108fc4afa6a05d9303c6ba68a128a55705d415985832fdeaae6c8e1
CTX 9110e84d1b1f199a2692119edc96132658f09da7c623efcec712537a3d94c0bf
CTX 5d7e352ec94ae5797fdb377dc1551150721adf15bd26a8efc2fcaad56881fa9e
CTX 62462c28f30ae1ceaca93c345cf243b73f542e2074a705bd2643bb9f7cc79bb6
CTX e7091ea6e232df0f9ad0d6cf502327876d82207abf2115cdacf6d5a48f6c1879
CTX a65b115f0f8b3cb3c59d15dd8c769bc014795a1837f3901b5845eb491adfefe0
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
CTX 97b1fa30a12fc1f65ba22905031539971a10f2f36c321bb51331cdefb39e3964
CTX c7ef079994f5b69b2edd83a71ef549971ee93f44eac3938fcdd61d01fa71799d
CTX a3a8091c4c48aa9ed263ff0749df95d44fef6a0bb578ec69456aa5408ae32c7a
CTX f08ad7ba8921287e3bbee31b767be06a0e705c864a769137df28292283ea81a2
CTX 480241b44d9921cdbec1bc28dc1fda114bd8e5217ac9d8ebafa720e9da4f9ace
CTX 231cc949e5b96fe76ffc21063fddc83a6b8679c00d35e09576a875305bed5f36
CTX ed242c8900dd1fa965bc950dfce09b132263a1eef52dd6888c309f5a7d712826

Vector 13
Key1 271828182845904523536028747135266249775724709369995957496696762 7
Key2 314159265358979323846264338327950288419716939937510582097494459 2
Data Unit Sequence Number ffffffff

PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX bf53d2dade78e822a4d949a9bc6766b01b06a8ef70d26748c6a7fc36d80ae4c5
CTX 520f7c4ab0ac8544424fa405162fef5a6b7f229498063618d39f0003cb5fb8d1
CTX c86b643497da1ff945c8d3bedeca4f479702a7a735f043ddb1d6aaade3c4a0ac
CTX 7ca7f3fa5279bef56f82cd7a2f38672e824814e10700300a055e1630b8f1cb0e
CTX 919f5e942010a416e2bf48cb46993d3cb6a51c19bacf864785a00bc2ecff15d3
CTX 50875b246ed53e68be6f55bd7e05cfc2b2ed6432198a6444b6d8c247fab941f5
CTX 69768b5c429366f1d3f00f0345b96123d56204c01c63b22ce78baf116e525ed9
CTX 0fdea39fa469494d3866c31e05f295ff21fea8d4e6e13d67e47ce722e9698a1c
CTX 1048d68ebcde76b86fcf976eab8aa9790268b7068e017a8b9b749409514f1053
CTX 027fd16c3786ea1bac5f15cb79711ee2abe82f5cf8b13ae73030ef5b9e4457e7
CTX 5d1304f988d62dd6fc4b94ed38ba831da4b7634971b6cd8ec325d9c61c00f1df
CTX 73627ed3745a5e8489f3a95c69639c32cd6e1d537a85f75cc844726e8a72fc00
CTX 77ad22000f1d5078f6b866318c668f1ad03d5a5fced5219f2eabbd0aa5c0f460
CTX d183f04404a0d6f469558e81fab24a167905ab4c7878502ad3e38fdbe62a4155
CTX 6cec37325759533ce8f25f367c87bb5578d667ae93f9e2fd99bcbc5f2fbba88c
CTX f6516139420fcff3b7361d86322c4bd84c82f335abb152c4a93411373aaa8220

Vector 14
Key1 271828182845904523536028747135266249775724709369995957496696762 7
Key2 314159265358979323846264338327950288419716939937510582097494459 2
Data Unit Sequence Number ffffffffff

PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 64497e5a831e4a932c09be3e5393376daa599548b816031d224bbf50a818ed23
CTX 50eae7e96087c8a0db51ad290bd00c1ac1620857635bf246c176ab463be30b80
CTX 8da548081ac847b158e1264be25bb0910bbc92647108089415d45fab1b3d2604
CTX e8a8eff1ae4020cfa39936b66827b23f371b92200be90251e6d73c5f86de5fd4
CTX a950781933d79a28272b782a2ec313efdfcc0628f43d744c2dc2ff3dcb66999b
CTX 50c7ca895b0c64791eeaa5f29499fb1c026f84ce5b5c72ba1083cddb5ce45434
CTX 631665c333b60b11593fb253c5179a2c8db813782a004856a1653011e93fb6d8
CTX 76c18366dd8683f53412c0c180f9c848592d593f8609ca736317d356e13e2bff
CTX 3a9f59cd9aeb19cd482593d8c46128bb32423b37a9adfb482b99453fbe25a41b
CTX f6feb4aa0bef5ed24bf73c762978025482c13115e4015aac992e5613a3b5c2f6
CTX 85b84795cb6e9b2656d8c88157e52c42f978d8634c43d06fea928f2822e465aa
CTX 6576e9bf419384506cc3ce3c54ac1a6f67dc66f3b30191e698380bc999b05abc
CTX e19dc0c6dcc2dd001ec535ba18deb2df1a101023108318c75dc98611a09dc48a
CTX 0acdec676fabdf222f07e026f059b672b56e5cbc8e1d21bbd867dd9272120546
CTX 81d70ea737134cdfce93b6f82ae22423274e58a0821cc5502e2d0ab4585e94de
CTX 6975be5e0b4efce51cd3e70c25a1fbbbd609d273ad5b0d59631c531f6a0a57b9
```

XTS-AES-128 applied for a data unit that is not a multiple of 16 bytesof 512 bytes

```
Vector 15
Key1 fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0
Key2 bfbebdbcbbbab9b8b7b6b5b4b3b2b1b0
Data unit sequence number 9a78563412

PTX 000102030405060708090a0b0c0d0e0f10
CTX 6c1625db4671522d3d7599601de7ca09ed

Vector 16
Key1 fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0
Key2 bfbebdbcbbbab9b8b7b6b5b4b3b2b1b0
Data unit sequence number 9a78563412

PTX 000102030405060708090a0b0c0d0e0f1011
CTX d069444b7a7e0cab09e24447d24deb1fedbf

Vector 17
Key1 fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0
Key2 bfbebdbcbbbab9b8b7b6b5b4b3b2b1b0
Data unit sequence number 9a78563412

PTX 000102030405060708090a0b0c0d0e0f101112
CTX e5df1351c0544ba1350b3363cd8ef4beedbf9d

Vector 18
Key1 fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0
Key2 bfbebdbcbbbab9b8b7b6b5b4b3b2b1b0
Data unit sequence number 9a78563412

PTX 000102030405060708090a0b0c0d0e0f10111213
CTX 9d84c813f719aa2c7be3f66171c7c5c2edbf9dac

Vector 19
Key1 e0e1e2e3e4e5e6e7e8e9eaebecedeeef
Key2 c0c1c2c3c4c5c6c7c8c9cacbcccdcecf
Data unit sequence number 21436587a9

PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 38b45812ef43a05bd957e545907e223b954ab4aaf088303ad910eadf14b42be6
CTX 8b2461149d8c8ba85f992be970bc621f1b06573f63e867bf5875acafa04e42cc
CTX bd7bd3c2a0fb1fff791ec5ec36c66ae4ac1e806d81fbf709dbe29e471fad3854
CTX 9c8e66f5345d7c1eb94f405d1ec785cc6f6a68f6254dd8339f9d84057e01a177
CTX 41990482999516b5611a38f41bb6478e6f173f320805dd71b1932fc333cb9ee3
CTX 9936beea9ad96fa10fb4112b901734ddad40bc1878995f8e11aee7d141a2f5d4
CTX 8b7a4e1e7f0b2c04830e69a4fd1378411c2f287edf48c6c4e5c247a19680f7fe
CTX 41cefbd49b582106e3616cbbe4dfb2344b2ae9519391f3e0fb4922254b1d6d2d
CTX 19c6d4d537b3a26f3bcc51588b32f3eca0829b6a5ac72578fb814fb43cf80d64
CTX a233e3f997a3f02683342f2b33d25b492536b93becb2f5e1a8b82f5b88334272
CTX 9e8ae09d16938841a21a97fb543eea3bbff59f13c1a18449e398701c1ad51648
CTX 346cbc04c27bb2da3b93a1372ccae548fb53bee476f9e9c91773b1bb19828394
CTX d55d3e1a20ed69113a860b6829ffa847224604435070221b257e8dff783615d2
CTX cae4803a93aa4334ab482a0afac9c0aeda70b45a481df5dec5df8cc0f423c77a
CTX 5fd46cd312021d4b438862419a791be03bb4d97c0e59578542531ba466a83baf
CTX 92cefc151b5cc1611a167893819b63fb8a6b18e86de60290fa72b797b0ce59f3
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex C

(informative)

Pseudocode for XTS-AES-128 and XTS-AES-256 Encryption

C.1 Encryption of a data unit with a size that is a multiple of 16 bytes

```
#define GF_128_FDBK       0x87
#define AES_BLK_BYTES     16
void XEX_EncryptSector
    (
    AES_Key &k2,                    // key used for tweaking
    AES_Key &k1,                    // key used for "ECB" encryption
    u64b  S,                        // data unit number (64 bits)
    uint  N,                        // sector size, in bytes
    const u08b *pt,                 // plaintext sector  input data
          u08b *ct                  // ciphertext sector output data
    )
    {
    uint    i,j;                    // local counters
    u08b    T[AES_BLK_BYTES];       // tweak value
    u08b    x[AES_BLK_BYTES];       // local work value
    u08b    Cin,Cout;               // "carry" bits for LFSR shifting

    assert(N % AES_BLK_BYTES == 0); // data unit is multiple of 16 bytes

    for (j=0;j<AES_BLK_BYTES;j++)
        {                           // convert sector number to tweak plaintext
        T[j] = (u08b) (S & 0xFF);
        S    = S >> 8;              // also note that T[] is padded with zeroes
        }

    AES_ECB_Encrypt(k2,T);          // encrypt the tweak

    for (i=0;i<N;i+=AES_BLK_BYTES)  // now encrypt the data unit, AES_BLK_BYTES at a time
        {
        // merge the tweak into the input block
        for (j=0;j<AES_BLK_BYTES;j++)
            x[j] = pt[i+j] ^ T[j];

        // encrypt one block
        AES_ECB_Encrypt(k1,x);

        // merge the tweak into the output block
        for (j=0;j<AES_BLK_BYTES;j++)
            ct[i+j] = x[j] ^ T[j];

        // Multiply T by α
        Cin = 0;
        for (j=0;j<AES_BLK_BYTES;j++)
            {
            Cout =  (T[j] >> 7) & 1;
            T[j] = ((T[j] << 1) + Cin) & 0xFF;
            Cin  =  Cout;
            }
        if (Cout)
            T[0] ^= GF_128_FDBK;
        }
    }
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.2 Encryption of a data unit with a size that is not a multiple of 16 bytes

```
#define GF_128_FDBK       0x87
#define AES_BLK_BYTES     16

void XEX_EncryptSector
    (
    AES_Key &k2,                    // key used for generating sector "tweak"
    AES_Key &k1,                    // key used for "ECB" encryption
    u64b  S,                        // sector number (64 bits)
    uint  N,                        // sector size, in bytes
    const u08b *pt,                 //  plaintext sector  input data
          u08b *ct                  // ciphertext sector output data
    )
    {
    uint    i,j;                    // local counters
    u08b    T[AES_BLK_BYTES];       // tweak value
    u08b    x[AES_BLK_BYTES];       // local work value
    u08b    Cin,Cout;               // "carry" bits for LFSR shifting

    assert(N >= AES_BLK_BYTES);     // need at least a full AES block

    for (j=0;j<AES_BLK_BYTES;j++)
        {                           // convert sector number to tweak plaintext
        T[j] = (u08b) (S & 0xFF);
        S    = S >> 8;              // also note that T[] is padded with zeroes
        }

    AES_ECB_Encrypt(k2,T);          // encrypt the tweak
    for (i=0;i+AES_BLK_BYTES <= N;i+=AES_BLK_BYTES)
        {                           // now encrypt the sector data
        // merge the tweak into the input block
        for (j=0;j<AES_BLK_BYTES;j++)
            x[j] = pt[i+j] ^ T[j];

        // encrypt one block
        AES_ECB_Encrypt(k1,x);

        // merge the tweak into the output block
        for (j=0;j<AES_BLK_BYTES;j++)
            ct[i+j] = x[j] ^ T[j];

        // LFSR "shift" the tweak value for the next location
        Cin = 0;
        for (j=0;j<AES_BLK_BYTES;j++)
            {
            Cout =  (T[j] >> 7) & 1;
            T[j] = ((T[j] << 1) + Cin) & 0xFF;
            Cin  =  Cout;
            }
        if (Cout)
            T[0] ^= GF_128_FDBK;
        }
    if (i < N)                          // is there a final partial block to handle?
        {
        for (j=0;i+j<N;j++)
            {
            x[j] = pt[i+j] ^ T[j];      // copy in the final plaintext bytes
            ct[i+j] = ct[i+j-AES_BLK_BYTES]; // and copy out the final ciphertext bytes
            }
        for ( j=0 ;j<AES_BLK_BYTES;j++)   // "steal" ciphertext to complete the block
            x[j] = ct[i+j-AES_BLK_BYTES] ^ T[j];
        // encrypt the final block
        AES_ECB_Encrypt(k1,x);

        // merge the tweak into the output block
        for (j=0;j<AES_BLK_BYTES;j++)
            ct[i+j-AES_BLK_BYTES] = x[j] ^ T[j];
        }
    }
```

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex D

(informative)

Rationale and Design Choices

D.1 Purpose

This Annex provides some background material regarding design choices that were made in XTS-AES and the rationale behind these choices.

D.2 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 data layout or message formats of 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 by manipulating the ciphertext, if it is willing to pay the price of "randomizing" the previous plaintext block.

The XTS-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

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

encryption, the constraints above imply some inherent limitations on the level of security that can be achieved by such transform. As shown 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 plaintext and there is no built-in mechanism to detect alterations. 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).[5] 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 several 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.

D.3 Wide vs. Narrow Block Tweakable Encryption

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

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

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*.

[5] 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.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(Such a construct is called a "tweakable cipher" in the cryptographic literature. It was first defined formally by Liskov et al. in [LRW02].)
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, requiring buffering of at least one sector worth of intermediate results and at least two passes over the entire sector.[6] 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, one can 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.
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.)

D.4 The XEX Construction

D.4.1 General XEX transform

In [XEX04], Rogaway described a construction of a narrow-block tweakable cipher from a standard cipher such as AES. That construction works as follows: The tweakable cipher uses two keys, K1 and K2, both used as keys for the underlying cipher Enc(K, data)/Dec(K, data). Given a plaintext block P and the tweak value, the tweak is parsed as a pair (s,t) (s can be thought of as the sector number and t as the block number within the sector). The construction first computes a mask value T using the following expression:

$T = Enc(K2, s) \otimes \alpha^t$

where the multiplication is in $GF(2^n)$ (with n being the block-size of the underlying cipher) and α is a primitive element of $GF(2^n)$. Given plaintext P, ciphertext C is produced by the following formula:

$C = Enc(K1, P \oplus T) \oplus T$

Given ciphertext C, the plaintext P is produced by the following formula:

$P = Dec(K1, C \oplus T) \oplus T.$

D.4.2 Security of general XEX transform

The security analysis of generic XEX transform in [XEX04] shows that this mode is secure as long as the number of blocks that are encrypted under the same key is sufficiently smaller than the birthday bound value of $2^{n/2}$; some attacks become possible when the number of blocks approaches the $2^{n/2}$ value, where n is the block size in bits of the underlying block cipher.

[6] 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.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The attacker analyzed in [XEX04] 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), and the attackers goal is to determine which is the case. Rogaway proved in [XEX04, Theorem 8] that an attacker that makes at most q such queries cannot distinguish these two cases with advantage more than $4.5\ q^2/2^n + \varepsilon$ over a random guess (where ε is an error term that expresses the advantage of distinguishing the underlying cipher from a random permutation using q queries and n is the block size in bits of the underlying block cipher).

To explain the relevance of this analysis to the security of a real-world usage of the XTS-AES transform, we first argue that no realistic attacker would have more information than the attacker in the attack model that is described in the analysis. This follows from the fact that attacker in [XEX04] is assumed to be able to choose all the plaintext and ciphertext that is fed to the construction. Since the theorem [XEX04, Theorem 8] 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 XEX.

It follows that when analyzing the security of an application that uses the above scheme, we can think of the encryption as if it was done using a collection of truly random 128-bit permutations. 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 (plaintext) will be decrypted (encrypted) to a random value (cf. the randomizing attack from above).

In other words, the proof in [XEX04] implies that except for the "error term" of $4.5\ q^2/2^n + \varepsilon$, the only attacks that are possible against XEX are the ones that are inherent from the use of transparent encryption with the granularity of n-bit blocks, where n is the block size in bits of the underlying cipher.

Some attacks against XEX are possible when the number of blocks q approaches the birthday bound. For example, consider a known-plaintext attack where the attacker sees q tuples of tweak, plaintext, and ciphertext. For each such tuple $((s_i,t_i), P_i, C_i)$, denote by T_i the mask value that is computed from the tweak (s_i,t_i).

From the birthday bound it follows that when q approaches $2^{n/2}$, there is a non-negligible probability that for some i,j there is a collision of the following form:

$P_i \oplus T_i = P_j \oplus T_j$.

In this case it also holds that:

$C_i \oplus T_i = Enc(K1, P_i \oplus T_i) = Enc(K1, P_j \oplus T_j) = C_j \oplus T_j$. (Equation 1)

Summing these two equalities implies

$P_i \oplus C_i = P_j \oplus C_j$

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This can be used to distinguish XEX from a collection of truly random permutations. The attacker computes for all i the sum $S_i=P_i \oplus C_i$ and counts the number of pairs (i,j) for which $S_i=S_j$. The argument above implies that for any i,j, the probability that $S_i=S_j$ in ciphertext produced by XEX is roughly $2^{-n}+2^{-n}=2^{-n+1}$, where the first term is due to collision between i and j and the second term is due to equality $S_i=S_j$ without a collision. On the other hand, for truly random permutation the probability of $S_i=S_j$ is exactly 2^{-n}, and hence after observing roughly $2^{n/2}$ tuples $((s_i,t_i), P_i, C_i)$ it is possible to distinguish ciphertext produced by XEX from a random sequence with non-negligible probability.

Given a collision between i and j as above, the following approach shows how the attacker can use his ability to create legally encrypted data for position i and ability to modify ciphertext in position j to modify the ciphertext at j so it will decrypt to an arbitrary attacker-controlled value.

As above, the attacker begins by computing the sums $S_i=C_i \oplus P_i$ and uses any equality $S_i=S_j$ as an evidence of collision between i and j. Denote by $((s_i,t_i), P_i, C_i)$, $((s_j,t_j), P_j, C_j)$ the corresponding tweak, plaintext, and ciphertext values.

For some $\Delta \neq 0$, the attacker encrypts a new value $P'_i=P_i \oplus \Delta$ in position (s_i,t_i), observes the corresponding ciphertext C'_i, and replaces the ciphertext block C_j by:

$C'_j = C_j \oplus (C_i \oplus C'_i)$.

This new ciphertext block will be decrypted as $P'_j=P_j \oplus \Delta$. In other words, the attacker succeeded in "flipping" specific bits in plaintext corresponding to location j. To see this, observe that:

$C'_j \oplus T_j = C_j \oplus (C_i \oplus C'_i) \oplus T_j$ (Equation 2)
$= C'_i \oplus (C_i \oplus C_j) \oplus T_j$
$= C'_i \oplus (T_i \oplus T_j) \oplus T_j$ (follows from Equation 1)
$= C'_i \oplus T_i$

Therefore:

$Dec(K1, C'_j \oplus T_j) = Dec(K1, C'_i \oplus T_i)$

which implies that:

$P'_j = T_j \oplus Dec(K1, C'_j \oplus T_j)$
$= T_j \oplus Dec(K1, C'_i \oplus T_i)$ (follows from Equation 2)
$= (T_j \oplus T_i) \oplus [T_i \oplus Dec(K1, C'_i \oplus T_i)]$
$= (T_j \oplus T_i) \oplus P'_i$
$= (T_j \oplus T_i) \oplus (P_i \oplus \Delta)$
$= ((T_j \oplus T_i) \oplus P_i) \oplus \Delta$
$= P_j \oplus \Delta$.

D.4.3 XTS-AES as a specific instantiation of general XEX

The XTS-AES-128 and XTS-AES-256 transforms described in this standard are concrete instantiations of the XEX scheme with AES as the underlying block cipher, and thus using n=128 as the block length. Relative position information is used as a tweak in order to allow for copy or backup of data encrypted with XTS-AES-[128,256] without re-encryption. In contrast to the generic XEX construction described in [XEX04] that uses a single key, the XTC-AES-128 and XTS-AES-256 modes in this standard use separate keys for tweaking and encryption purposes. This separation is a specific example of separation of key usage by purpose and is considered a good security design practice (see [KEY-MGMT, part 1, Section 5.2]).

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The expression $4.5\ q^2/2^n$ is small enough as long as q is not much more than 2^{40}. The proof from [XEX04] yields strong security guarantee as long as the same key is not used to encrypt much more than a terabyte of data (which gives $q=2^{36}$ blocks). For this case, no attack can succeed with probability better than 2^{-53} (i.e., approximately one in eight quadrillion).

This security guarantee deteriorates as more data is encrypted under the same key. For example, when using the same key for a petabyte of data (2^{46} blocks), attacks such as in D.4.2 have success probability of 2^{37} (i.e., approximately eight in a trillion), and with exabyte of data the success probability is 2^{17} (i.e., approximately eight in a million).

The decision on the maximum amount of data to be encrypted with a single key should take into account the above calculations together with the practical implication of the described attack, e.g. ability of the attacker to modify plaintext of a specific block, where the position of this block may not be under attacker's control.

D.5 Sector-size which is not a multiple of 128 bits

The generic XEX transform as described in [XEX04] immediately implies a method for encrypting sectors that consist of an integral number of 128-bit blocks: apply the transform 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).

To encrypt a sector which length is not an integral number of 128-bit blocks, the standard uses the "ciphertext-stealing" technique similar to the one used for ECB mode (see [MM82, Fig. 2-22]). Namely, both XTS-AES-128 and XTS-AES-256 encrypt all the full blocks *except the last* (with different tweak values for each block), and then encrypt the last full block together with the remaining partial block using two application of the XTS-AES-blockEnc procedure described in 5.2.1 with two different tweak values, as described in 5.2.2

D.6 Miscellaneous

Following are general remarks about appropriate use of the XTS-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 XTS-AES-[128,256] be chosen at random (or from a "cryptographically strong" pseudo-random source). Indeed, all security guarantees (including the security claims of the theorem from [XEX04]) 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].

- Use of a single cryptographic key for more than a few hundreds terabytes of data opens possibility of attacks, as described in D.4.3. We note that limitation on size of data encrypted with a single key is not unique to this standard. It comes directly from the fact that AES has block size of 128 bits and is not mitigated by using AES with a 256-bit key.

Copyright © 2006 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.