The XTS-AES Tweakable Block Cipher
An Extract from IEEE Std 1619-2007

Extracted from IEEE Std 1619-2007, published 18 April 2008.

This special IEEE copyrighted PDF is being created to allow NIST to submit IEEE Std 1619-2007 XTS-AES encryption algorithm for consideration as an Approved Mode of Operation under NIST FIPS 140-2 This document is made available for public review only for a period of ninety (90) days (June 5, 2008 through September 3, 2008). Copying, cutting and pasting, and/or redistributing electronically or otherwise of this document is strictly prohibited.

Introduction

This document was extracted from IEEE Std 1619-2007, IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. This document contains description of the XTS-AES transform. Please refer to the full standard documentation for other information, including motivation, key export/import, and test vectors.

XTS-AES is a tweakable block cipher designed for encryption of sector-based storage. XTS-AES 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 Rogaway [B10][a]. The XTS-AES addresses threats such as copy-and-paste attack, while allowing parallelization and pipelining in cipher implementations.

Notice to users

Laws and regulations

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

Copyrights

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

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata.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

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

[a] The numbers in brackets correspond to those of the bibliography in Annex A.

Errata

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

Interpretations

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

Patents

Attention is called to the possibility that implementation of this IEEE Std 1619-2007 may require use of subject matter covered by patent rights. By publication of this IEEE Std 1619-2007, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this IEEE Std 1619-2007 are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants from IEEE Std 1619-2007

The Security in Storage Working Group operated under the following sponsorship:

Sponsor: **John L. Cole**
Co-Sponsor: **Curtis Anderson**

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the Security in Storage Working Group had the following officers:

Matthew V. Ball, *Chair*
Eric A. Hibbard, *Vice-chair*
James P. Hughes, *Past chair*
Fabio Maino, *Secretary*

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the P1619 Task Group had the following membership:

Serge Plotkin, *Task Group Chair and Technical Editor*

Gideon Avida
Matthew V. Ball
David L. Black
Russel S. Dietz
Robert C. Elliott
Hal Finney
John Geldman
Robert W. Griffin
Cyril Guyot
Shai Halevi
Laszlo Hars
Larry D. Hofer
Walter A. Hubis
James P. Hughes
Glen Jaquette
Curt Kolovson
Robert A. Lockhart
Fabio R. Maino
Charlie Martin
David A. McGrew
Dalit Naor
Landon Curt Noll
Jim Norton
Scott Painter
David B. Sheehy
Robert N. Snively
Douglas L. Whiting

This "***Extract of IEEE Std 1619-2007***" was edited by Serge Plotkin (*past vice-chair*), Shai Halevi, and Dalit Naor.

Special thanks to Douglas L. Whiting, Brian Gladman, and Robert C. Elliott.

IEEE Std 1619-2007 Balloters

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

Curtis C. Anderson
Danilo Antonelli
Gideon Avida
Matthew V. Ball
Brian A. Berg
Massimo Cardaci
Juan C. Carreon
Keith Chow
John L. Cole
Roger Cummings
Geoffrey Darnton
Russell S. Dietz
Carlo Donati
Robert C. Elliott
Yaacov Fenster
John Geldman
Robert W. Griffin
Randall Groves
Laszlo Hars
Eric A. Hibbard
Werner Hoelzl
Larry D. Hofer
Stuart Holoman
Walter A. Hubis
Raj Jain
Piotr Karocki
Kenneth Lang
Daniel G. Levesque
Robert A. Lockhart
Fabio R. Maino
Edward McCall
Michael Newman
Charles K. Ngethe
Landon Curt Noll
Serge Plotkin
Ulrich Pohl
Jose Puthenkulam
Michael D. Rush
Randall M. Safier
Stephen Schwarm
David B. Sheehy
Robert N. Snively
Thomas Starai
Walter Struppler
John Vergis
Oren Yuen
Paolo Zangher

CONTENTS

IEEE Std 1619-2007

IMPORTANT NOTICE: This standard is not intended to assure safety, security, health, or environmental protection in all circumstances. Implementers of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.
This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading "Important Notice" or "Important Notices and Disclaimers Concerning IEEE Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/disclaimer.html.

1. Overview

1.1 Scope

This standard specifies elements of an architecture for cryptographic protection of data on block-oriented storage devices, describing the methods, algorithms, and modes of data protection to be used.

1.2 Purpose

This standard defines specific elements of an architecture for cryptographically protecting data stored in constant length blocks. Specification of such a mechanism provides an additional and improved tool for implementation of secure and interoperable protection of data residing in storage.

1.3 Related work

The formal definition of the security goal of a tweakable block-cipher can be attributed to Liskov, Rivest, and Wagner [B5][1], 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 (see Schroeppel [B11]).

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). 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 (AES).[2]

[1] The numbers in brackets correspond to those of the bibliography in Annex A.

[2] FIPS publications are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22661, USA. FIPS-197 is also available on-line from http://csrc.nist.gov/publications/fips/index.

3. Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition* [B4] should be referenced for terms not defined in this clause.

key scope: Data encrypted by a particular key, divided into equal-sized data units. The key scope is identified by three non-negative integers: tweak value corresponding to the first data unit, the data unit size, and the length of the data.

NOTE—See 4.3.1.[3]

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

3.1 Acronyms and abbreviations

AES	advanced encryption standard
Base64	encoding according to IETF RFC 3548 [B12]
DTD	document type definition
FIPS	Federal Information Processing Standard
GF	Galois field (see Menezes et. al. [B6])
LBA	logical block address
XML	extensible markup language
XTS	XEX encryption mode with tweak and ciphertext stealing

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. Hexadecimal numbers are represented by a string of one or more hexadecimal characters followed by a subscript 16.

[3] Notes in text, tables, and figures are given for information only, and do not contain requirements needed to implement the standard.

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 (see Menezes et. al. [B6])
α	A primitive element of GF(2^{128}) that corresponds to polynomial x (i.e., $0000\ldots010_2$), where GF stands for Galois Field (see Menezes et. al. [B6])
•	Assignment of a value to a variable
|	Concatenation (e.g., if $K1 = 001_2$ and $K2 = 101010_2$, then $K1\|K2 = 001101010_2$)
//	Start of a comment. Comment ends at end of line
$\lfloor x \rfloor$	Floor of x (e.g., $\lfloor 7/3 \rfloor = 2$)

4.3 Special definitions

4.3.1 Data unit: Within IEEE Std 1619, 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 or logical block on the storage device.

5. XTS-AES transform

5.1 Data units and tweaks

This standard applies to encryption of a data stream divided into consecutive equal-size data units, where the data stream refers to the information that has to be encrypted and stored on the storage device. Information that is not to be encrypted is considered to be outside of the data stream.

The data unit size shall be at least 128 bits. Data unit should be divided into 128-bit blocks. Last part of the data unit might be shorter than 128 bits. The number of 128-bit blocks in the data unit shall not exceed $2^{128} – 2$. The number of 128-bit blocks should not exceed 2^{20}.[4] Each data unit is assigned a tweak value that is a non-negative integer. The tweak values are assigned consecutively, starting from an arbitrary non-negative integer. When encrypting a tweak value using AES, the tweak is first converted into a little-endian byte array. For example, tweak value $123456789a_{16}$ corresponds to byte array $9a_{16}, 78_{16}, 56_{16}, 34_{16}, 12_{16}$.

The mapping between the data unit and the transfer, placement, and composition of data on the storage device is beyond the scope of this standard. Devices compliant with this standard should include documentation describing this mapping. In particular, a single data unit does not necessarily correspond to a single logical block on the storage device. For example, several logical blocks might correspond to a single data unit. Data stream, as used in this standard, does not necessarily refer to all of the bits sent to be

[4] Previous two sentences are not contradicting each other. First sentence states the hard limit, while the second one strongly suggests to keep the value below the second, significantly lower limit.

stored in the storage device. For example, if only part of a logical block is encrypted, only the encrypted bytes are viewed as the data stream, i.e., input to the encryption algorithm in this standard.

5.2 Multiplication by a primitive element α

The encryption procedure (see 5.3) and decryption procedure (see 5.4) use multiplication of a 16-byte value (the 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 a byte 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
byte array $a_0[k]$, $k = 0,1,...,15$
Output: byte 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:

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

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

5.3 XTS-AES encryption procedure

5.3.1 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 Equation (1).

$$C \leftarrow \text{XTS-AES-blockEnc}(Key, P, i, j) \qquad (1)$$

where

- Key is the 256 or 512 bit XTS-AES key
- P is a block of 128 bits (i.e., the plaintext)
- i is the value of the 128-bit tweak (see 5.1)
- 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 (see Figure 1):

1) $T \leftarrow \text{AES-enc}(Key_2\,,\ i) \otimes \alpha^j$
2) $PP \leftarrow P \oplus T$
3) $CC \leftarrow \text{AES-enc}(\text{Key}_1\,,\ PP)$
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 the primitive element defined in 4.2 (see 5.2).

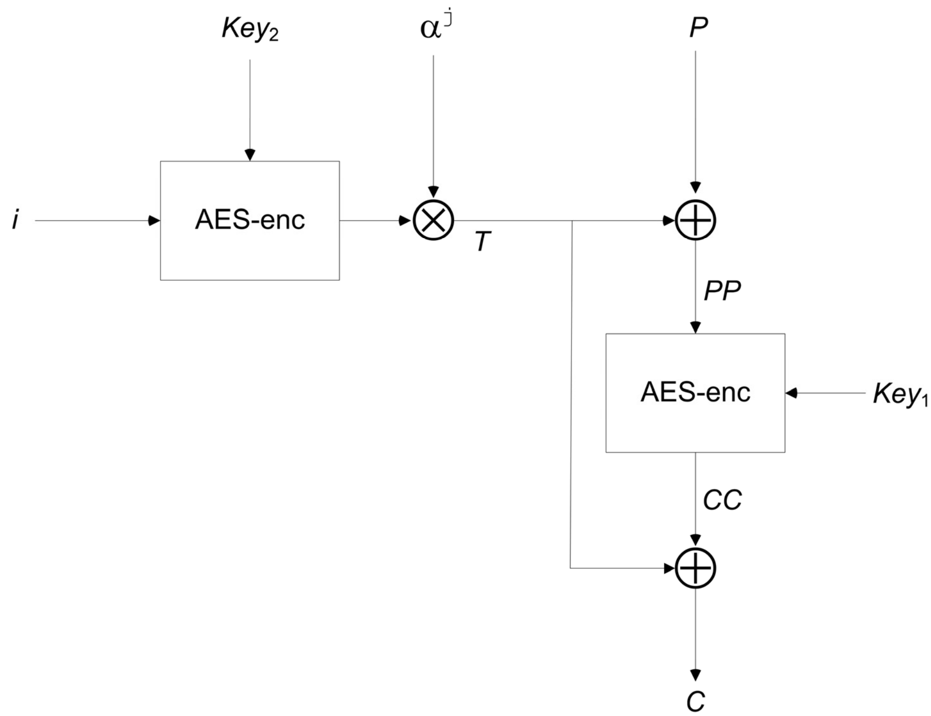

Figure 1— Diagram of XTS-AES blockEnc procedure

5.3.2 XTS-AES encryption of a data unit

The XTS-AES encryption procedure for a data unit of plaintext of 128 or more bits is modeled with Equation (2).

$$C \leftarrow \text{XTS-AES-Enc}(\text{Key}, P, i) \tag{2}$$

where

- *Key* is the 256 or 512 bit XTS-AES key
- *P* is the plaintext
- *i* is the value of the 128-bit tweak (see 5.1)
- *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, as follows:

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

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 each exactly 128 bits long, and the last block P_m is between 0 and 127 bits long (P_m could be empty, i.e., 0 bits long). 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 C is then computed by the following or an equivalent sequence of steps:

```
1) for q ← 0 to m-2 do
   a)    Cq ← XTS-AES-blockEnc(Key, Pq, i, q)
2) b ← bit-size of Pm
3) if b = 0 then do
   a)    Cm-1 ← XTS-AES-blockEnc(Key, Pm-1, i, m-1)
   b)    Cm ← empty
```

4) else do
   a) $CC \leftarrow$ XTS-AES-blockEnc(Key, P_{m-1}, i, $m-1$)
   b) $C_m \leftarrow$ first b bits of CC
   c) $CP \leftarrow$ last ($128-b$) bits of CC
   d) $PP \leftarrow P_m \mid CP$
   e) $C_{m-1} \leftarrow$ XTS-AES-blockEnc(Key, PP, i, m)
5) $C \leftarrow C_0 \mid \ldots \mid C_{m-1} \mid C_m$

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.

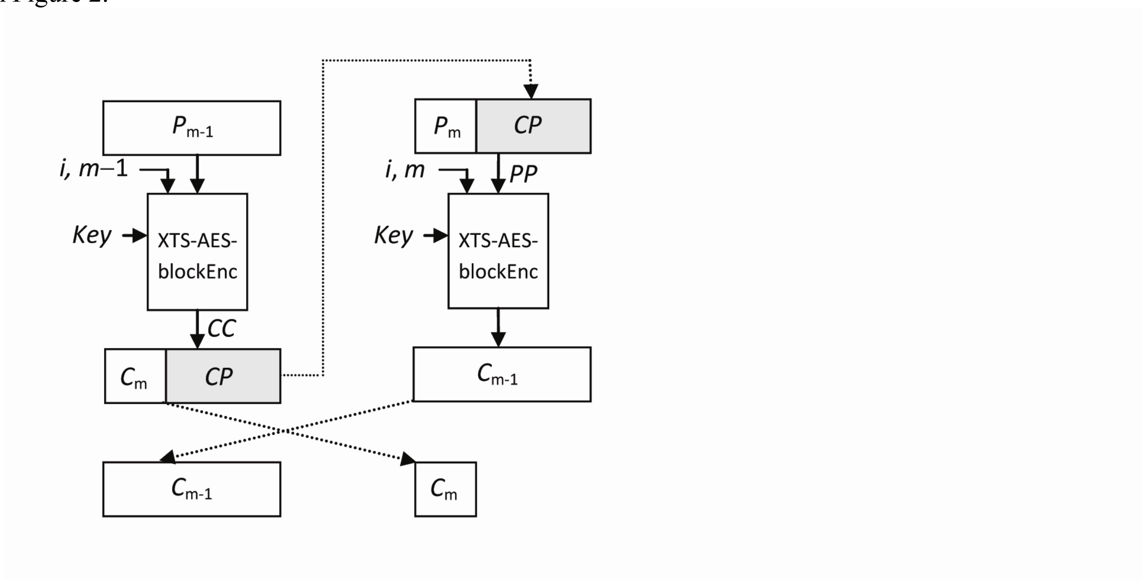

Figure 2—XTS-AES encryption of last two blocks when last block is 1 to 127 bits

5.4 XTS-AES decryption procedure

5.4.1 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 Equation (3).

$$P \leftarrow \text{XTS-AES-blockDec}(Key, \mathrm{C}, i, j) \tag{3}$$

where

Key	is the 256 or 512-bit XTS-AES key
C	the 128-bit block of ciphertext
i	is the value of the 128-bit tweak (see 5.1)
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 (see Figure 3):

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 the primitive element defined in 4.2 (see 5.2).

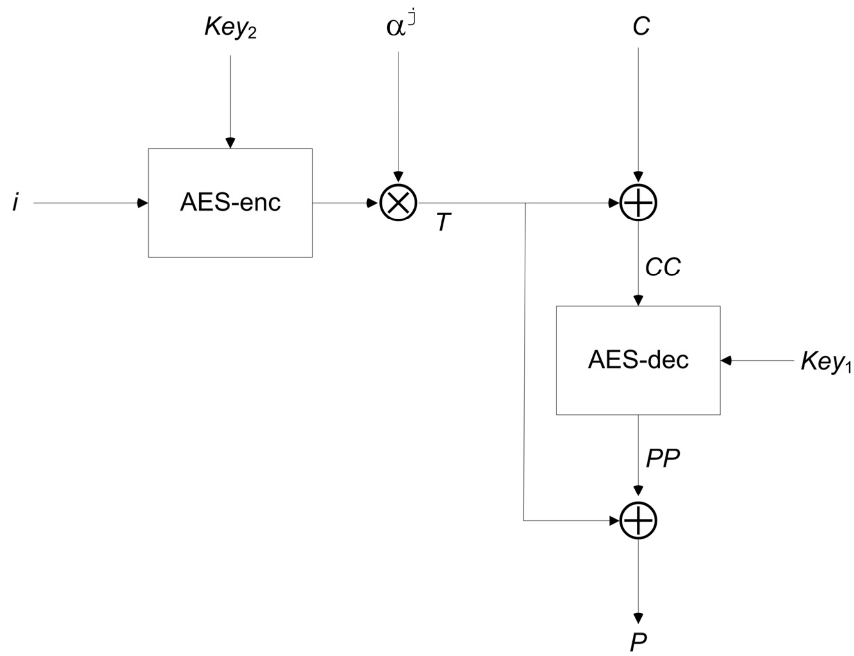

Figure 3—Diagram of XTS-AES blockDec procedure

5.4.2 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 Equation (4).

$$P \leftarrow \text{XTS-AES-Dec}(Key, C, i) \tag{4}$$

where

Key	is the 256 or 512-bit XTS-AES key
C	is the ciphertext corresponding to the data unit
i	is the value of the 128-bit tweak (see 5.1)
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 as follows:

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

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 each exactly 128 bits long, and the last block C_m is between 0 and 127 bits long (C_m could be empty, i.e., 0 bits long). 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 P is then computed by the following or an equivalent sequence of steps:

```
1)  for q ← 0 to m-2 do
     a)     Pq ← XTS-AES-blockDec(Key, Cq, i, q)
2)  b ← bit-size of Cm
```

3) if $b = 0$ then do
  b) $P_{m-1} \leftarrow$ XTS-AES-blockDec(Key, C_{m-1}, i, m-1)
  c) $P_m \leftarrow$ empty
4) else do
  d) $PP \leftarrow$ XTS-AES-blockDec(Key, C_{m-1}, i, m)
  e) $P_m \leftarrow$ first b bits of PP
  f) $CP \leftarrow$ last (128-b) bits of PP
  g) $CC \leftarrow C_m \mid CP$
  h) $P_{m-1} \leftarrow$ XTS-AES-blockDec(Key, CC, i, m-1)
5) $P \leftarrow P_0 \mid \ldots \mid P_{m-1} \mid P_m$

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

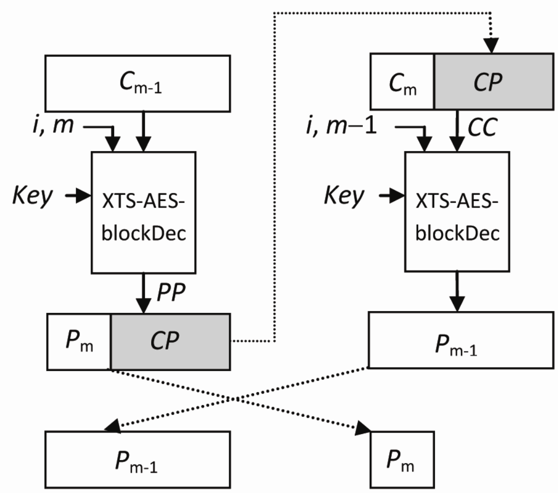

Figure 4—XTS-AES decryption of last two blocks when last block is 1 to 127 bits

6. Using XTS-AES-128 and XTS-AES-256 for encryption of storage

The encryption and decryption procedures described in 5.3 and 5.4 use AES as the basic building block. If the XTS-AES key consists of 256 bits, the procedures use 128-bit AES; if the XTS-AES key consists of 512 bits, the procedures use 256-bit AES. For completeness, the first 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.

Key scope defines the range of data encrypted with a single XTS-AES key. The key scope is represented by the following three values:

a) Value of the tweak associated with the first data unit in the sequence of data units encrypted by this key
b) The size in bits of each data unit
c) The number of units to be encrypted/decrypted under the control of this key.

An implementation compliant with this standard may or may not support multiple data unit sizes.

In an application of this standard to sector-level encryption of a disk, the data unit typically corresponds to a logical block, the key scope typically includes a range of consecutive logical blocks on the disk, and the tweak value associated with the first data unit in the scope typically corresponds to the Logical Block Address (LBA) associated with the first logical block in the range.

An XTS-AES key shall not be associated with more than one key scope.

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

Annex A

(informative)

Bibliography

[B1] Halevi, S. and Rogaway, P., “A tweakable enciphering mode.” In Advances in Cryptology—CRYPTO ’03. Lecture Notes in Computer Science, vol. 2729, pp 482–499. Springer-Verlag, 2003.

[B2] Halevi, S. and Rogaway, P., *A parallelizable enciphering mode.* The RSA conference—Cryptographer’s track, RSA-CT ’04. Lecture Notes in Computer Science, vol. 2964, pp 292–304. Springer-Verlag, 2004.

[B3] Halevi, S., “EME*: extending EME to handle arbitrary-length messages with associated data,” INDOCRYPT 2004, Lecture Notes in Computer Science, vol. 3348, pp 315–327. Springer-Verlag, 2004.

[B4] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms,* Seventh Edition.

[B5] Liskov, M., Rivest, R., and Wagner, D., *Tweakable block ciphers.* In Advances in Cryptology—CRYPTO ’02. Lecture Notes in Computer Science, vol 2442, pp 31–46. Springer-Verlag, 2002.

[B6] Menezes, A., Oorshot, P., and Vanstone, S., *Handbook of Applied Cryptography,* CRC Press, 1997.

[B7] Meyer, C. H. and Matyas, S. M., *Cryptography: a New Dimension in Computer Data Security.* John Wiley & Sons, 1982.

[B8] Naor, M. and Reingold, O., *A pseudo-random encryption mode.* Manuscript. Available online from http://www.wisdom.weizmann.ac.il/~naor/PAPERS/nr-mode.ps.

[B9] 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

[B10] Rogaway, P., *Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC.* Advances in Cryptology—Asiacrypt 2004. Lecture Notes in Computer Science, vol. 3329, pp 16–31. Springer-Verlag, 2004.

[B11] Schroeppel, R., *The Hasty Pudding Cipher.* The first AES conference, NIST, 1998.
Available from http://www.cs.arizona.edu/~rcs/hpc.
(See http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/aes1conf.htm.)

[B12] The Base16, Base32, and Base64 Data Encodings. IETF Network Working Group, July 2003. http://www.ietf.org/rfc/rfc3548.txt

[B13] XML Encryption Syntax and Processing. W3C.
http://www.w3.org/TR/xmlenc-core

[B14] XML Key Management Specification (XKMS). W3C.
http://www.w3.org/TR/xkms

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 XTS_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;
        }
    }
```

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 XTS_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<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];
        }
    }
```

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, 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 (CTR) mode or cipher block chaining (CBC) mode, deriving the IV from the sector number.

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

— For CBC, an adversary 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 adversary 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.

— For CBC, an adversary can flip any bit of the plaintext by flipping the corresponding ciphertext bit of the previous block, with the side-effect of "randomizing" the previous 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 encryption, the constraints above imply some inherent limitations on the level of security that can be achieved by such transform. As shown in the paragraphs that follow, 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, any ciphertext (original or modified by adversary) 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, encrypting the plaintext twice with the same key and the same position will yield the same ciphertext. Moreover, since there is no chaining, an adversary can "mix and match" ciphertext units and get the same "mix and match" of their corresponding plaintext units. (Namely, if $C_0C_1\ldots C_m$ is encryption of $P_0P_1\ldots P_m$ and $C'_0C'_1\ldots C'_m$ is encryption of $P'_0P'_1\ldots P'_m$ then $C_0C'_1\ldots C_m$ is encryption of $P_0P'_1\ldots P_m$.)

The above "mix and match" weakness 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 previously, even the best implementation of encryption by a sector-level storage device leaves several vulnerabilities. Three of these vulnerabilities are illustrated as follows:

- **Traffic analysis.** Consider an adversary that is able to passively observe the communication between the encrypting device and the disk. Since encryption is deterministic, this adversary 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 adversary in mounting an attack based on traffic analysis.
- **Replay.** An adversary with read/write access to the encrypted disk can observe when a certain sector changes on the disk and then reset it to any one of its previous values. (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 adversary 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 adversary.

When using transparent encryption, one must therefore address these vulnerabilities by means outside the scope of this standard.

D.3 Wide vs. narrow block tweakable encryption

In light of the previous discussion, the required interfaces of the transform are encryption and decryption routines as shown in Equation (5):

$$C = \mathrm{Enc}(K, P, i) \text{ and } P = \mathrm{Dec}(K, C, i) \tag{5}$$

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

[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.

The best security that one can hope for with such transform is that it looks to an adversary 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 [B5].

Several constructions that achieve these properties exist in the cryptographic literature (e.g., see Halevi et. al. [B1], [B2], [B3], and a construction based on Naor et. al. [B8]). 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 adversary 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 D.2 are now possible with finer granularity. Namely, whereas the adversary against a wide-block encryption scheme can do traffic analysis or replay with granularity of one sector, the adversary against a narrow-block encryption scheme can work with granularity of 128-bit blocks. Still, the consensus in the P1619 workgroup was that the added efficiency warrants this additional risk. Since these risks exist even with wide-block encryption—albeit with a coarser granularity—one would still 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 XEX construction

D.4.1 General XEX transform

In [B10], 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 Equation (6):

$$\mathrm{T} = \mathrm{Enc}(\mathrm{K2}, \mathrm{s}) \otimes \alpha^{\mathrm{t}} \qquad (6)$$

where
the multiplication is in GF(2^n) (with n being the block-size of the underlying cipher)
α is a primitive element of GF(2^n)

Given plaintext P, ciphertext C is produced by Equation (7):

$$C = \mathrm{Enc}(\mathrm{K1},\ P \oplus T) \oplus T \qquad (7)$$

Given ciphertext C, the plaintext P is produced Equation (8):

$$P = \mathrm{Dec}(\mathrm{K1}, C \oplus T) \oplus T \qquad (8)$$

[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.

D.4.2 Security of general XEX transform

The security analysis of generic XEX transform in Rogaway [B10] 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 $2^{n/2}$, where n is the block size in bits of the underlying block cipher. Some attacks become possible when the number of blocks approaches the $2^{n/2}$ value.

The adversary analyzed in Rogaway [B10] 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 adversary's goal is to determine which is the case. Rogaway proved in [B10], Theorem 8 that an adversary 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, the first argument is that no realistic adversary would have more information than the adversary in the attack model that is described in the analysis. This follows from the fact that adversary in Rogaway [B10] is assumed to be able to choose all the plaintext and ciphertext that is fed to the construction. Since the theorem (Rogaway [B10], Theorem 8) says that no adversary in that model can distinguish the construction from a collection of random permutations, it follows that no realistic adversary can distinguish between these cases with any significant advantage. This, in turn, means that an attack would be just as successful against a collection of truly random permutations, one per each 128-bit block, as it would be against XEX.

It follows that when analyzing the security of an application that uses the above scheme, one 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 adversary 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 Rogaway [B10] 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 adversary 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 form shown in Equation (9):

$$P_i \oplus T_i = P_j \oplus T_j. \tag{9}$$

In this case, it also holds that [see Equation (10)]:

$$C_i \oplus T_i = \mathrm{Enc}(\mathrm{K1}, P_i \oplus T_i) = \mathrm{Enc}(\mathrm{K1}, P_j \oplus T_j) = C_j \oplus T_j. \tag{10}$$

Summing these two equalities implies

$$P_i \oplus C_i = P_j \oplus C_j$$

This can be used to distinguish XEX from a collection of truly random permutations. The adversary 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 adversary 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 adversary-controlled value.

As above, the adversary 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 adversary 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 adversary succeeded in "flipping" specific bits in plaintext corresponding to location j. To see this, observe Equation (11):

$$\begin{aligned} C'_j \oplus T_j &= C_j \oplus (C_i \oplus C'_i) \oplus T_j \qquad (11)\\ &= C'_i \oplus (C_i \oplus C_j) \oplus T_j \\ &= C'_i \oplus (T_i \oplus T_j) \oplus T_j \qquad \text{[follows from Equation (10)]}\\ &= C'_i \oplus T_i \end{aligned}$$

Therefore:

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

which implies that:

$$\begin{aligned} P'_j &= T_j \oplus \mathrm{Dec}(K1, C'_j \oplus T_j) \\ &= T_j \oplus \mathrm{Dec}(K1, C'_i \oplus T_i) \qquad \text{[follows from Equation (11)]}\\ &= (T_j \oplus T_i) \oplus [T_i \oplus \mathrm{Dec}(K1, C'_i \oplus T_i)] \\ &= (T_j \oplus T_i) \oplus P'_i \\ &= (T_j \oplus T_i) \oplus (P_i \oplus \Delta) \\ &= ((T_j \oplus T_i) \oplus P_i) \oplus \Delta \\ &= P_j \oplus \Delta. \end{aligned}$$

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. A data unit sequence number (i.e., relative position) is used as a tweak in order to allow for copy or backup of a key scope or partial key scope of data encrypted with XTS-AES-[128,256] without re-encryption. In contrast to the generic XEX construction described in Rogaway [B10] that uses a single key, the XTS-

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 NIST Key Management Guidelines [B9], part 1, Section 5.2).

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 Rogaway [B10] 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, attacks such as in D.4.2 have success probability of at most approximately 2^{-37} (i.e., approximately eight in a trillion), and with exabyte, of data the success probability is at most approximately 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 adversary to modify plaintext of a specific block, where the position of this block may not be under adversary's control).

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

The generic XEX transform as described in Rogaway [B10] 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 with a length that 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 Meyer et. al. [B7], Figure 2-22). Namely, both XTS-AES-128 and XTS-AES-256 encrypt all the full blocks except the last full block (with different tweak values for each block), and then encrypt the last full block together with the remaining partial block using two applications of the XTS-AES-blockEnc procedure described in 5.3.1 with two different tweak values, as described in 5.3.2.

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 Rogaway [B10]) 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 NIST Key Management Guidelines [B9].

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