IEEE

1619.1™

IEEE Standard for Authenticated Encryption with Length Expansion for Storage Devices

IEEE Computer Society

Sponsored by the
Storage Systems Standards Committee
and
Information Assurance Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

16 May 2008

IEEE Std 1619.1™-2007

Recognized as an
American National Standard (ANSI)

IEEE Std 1619.1™-2007

IEEE Standard for Authenticated Encryption with Length Expansion for Storage Devices

Sponsor
Information Assurance Standards Committee
and
Storage Systems Standards Committee
of the
IEEE Computer Society

Approved 15 April 2008

American National Standards Institute

Approved 5 December 2007

IEEE-SA Standards Board

Abstract: Cryptographic and data authentication procedures for storage devices that support length expansion, such as tape drives, are specified. Such procedures include the following cryptographic modes of operation for the AES block cipher: CCM, GCM, CBC-HMAC, and XTS-HMAC.

Keywords: authentication, CBC, CCM, cryptography, data storage, encryption, GCM, HMAC, security, tape drive, variable-length block, XTS

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-0-7381-5360-5 STD95753
Print: ISBN 978-0-7381-5361-2 STDPD95753

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

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

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

This introduction is not part of IEEE Std 1619.1-2007, IEEE Standard for Authenticated Encryption with Length Expansion for Storage Devices.

The problem of data storage protection has become increasingly important due to legislation that requires the protection of sensitive information. To address this issue, the Security in Storage Working Group (SISWG) is developing standards for the protection of information on storage media. This standard provides strong data protection by specifying encryption with authentication and length expansion.

This standard provides methods suitable for ensuring the privacy and integrity of stored data within applications requiring a high level of assurance. To this end, this standard specifies the Advanced Encryption Standard (AES) cipher as used in authenticated-encryption modes.

There are many modes of non-cryptographic attacks that are outside the scope of this standard. See B.1 for a discussion.

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 website 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 website at http://standards.ieee.org.

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 guide may require use of subject matter covered by patent rights. By publication of this guide, 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 guide 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

When work on this standard began, the Security in Storage Working Group had the following officers:

James P. Hughes, *Chair*

Serge Plotkin, *Vice-chair*

Fabio Maino, *Secretary*

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

John L. Cole, *Sponsor*

Curtis Anderson, *Co-Sponsor*

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*

Fabio Maino, *Secretary*

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the P1619.1 Task Group had the following membership:

Matthew V. Ball, *Chair, Technical Editor*

Gideon Avida
Matthew V. Ball
Mark Benedikt
Jon Buckingham
William G. Colvin
Russel S. Dietz
Robert Drennan
Hal Finney
John Geldman
Robert W. Griffin
Shai Halevi *(past technical editor)*
Laszlo Hars

Eric A. Hibbard
Larry D. Hofer
Jon Holdman
Walter A. Hubis
James P. Hughes *(past technical editor)*
Glen Jaquette *(past technical editor)*
Curt Kolovson
Robert A. Lockhart
Fabio R. Maino
Charlie Martin
Dalit Naor

Landon Curt Noll
Jim Norton
Scott Painter
David Peterson
Serge Plotkin
David B. Sheehy
Robert N. Snively
Alexander (Sandy) Stewart
Greg Unruh
Douglas L. Whiting
Christopher L. Williams
Mike Witkowski

Special thanks to Brian Gladman, David McGrew, Michael Torla, and Danh Tran

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

Danilo Antonelli
Matthew V. Ball
Rahul B. Bhushan
Juan C. Carreon
Lidong Chen
Keith Chow
Roger Cummings
Geoffrey Darnton
Russell S. Dietz
Thomas J. Dineen
Robert C. Elliott
Michael D. Geipel
John Geldman
Brian Goodman

Robert W. Griffin
Randall Groves
Eric A. Hibbard
Werner Hoelzl
Larry D. Hofer
Walter A. Hubis
Raj Jain
Robert A. Lockhart
William Lumpkins
Fabio R. Maino
Edward McCall
Michael Newman
Landon Curt Noll
Mark Overby

Serge Plotkin
Ulrich Pohl
Venkatesha Prasad
Michael D. Rush
David B. Sheehy
Avraham Shimor
Robert N. Snively
Thomas Starai
Walter Struppler
Joseph Tardo
Wen Tong
Thomas Tullia
Christopher L. Williams
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 5 December 2007, it had the following membership:

Steve M. Mills, *Chair*
Robert M. Grow, *Vice Chair*
Don Wright, *Past Chair*
Judith Gorman, *Secretary*

Richard DeBlasio
Alex Gelman
William R. Goldbach
Arnold M. Greenspan
Joanna N. Guenin
Julian Forster*
Kenneth S. Hanus
William B. Hopf

Richard H. Hulett
Hermann Koch
Joseph L. Koepfinger*
John Kulick
David J. Law
Glenn Parsons
Ronald C. Petersen
Tom A. Prevost

Narayanan Ramachandran
Greg Ratta
Robby Robson
Anne-Marie Sahazizian
Virginia C. Sulzberger
Malcolm V. Thaden
Richard L. Townsend
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Alan H. Cookson, *NIST Representative*

Lorraine Patsco
IEEE Standards Program Manager, Document Development

Michael Kipness
IEEE Standards Program Manager, Technical Program Development

CONTENTS

IEEE Standard for Authenticated Encryption with Length Expansion for Storage Devices

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/disclaimers.html.

1. Overview

1.1 Scope

This standard specifies requirements for cryptographic units that provide encryption and authentication for data contained within storage media. Full interchange requires additional format specifications (such as compression algorithms and physical data format) that are beyond the scope of this standard.

1.2 Purpose

This standard is suitable for encryption of data stored on tape because tape easily accommodates length-expanding ciphertext. In addition, this standard applies to other storage devices if these support storing extra metadata with each encrypted record. The algorithms of this standard are designed to ensure the confidentiality and integrity of stored data within systems requiring a high level of assurance.

1.3 Description of clauses and annexes

— Clause 1 provides an overview of this standard, including scope and purpose.

— Clause 2 lists the normative references that are essential for implementing this standard.

— Clause 3 gives definitions, acronyms, and abbreviations used in this standard.

— Clause 4 provides a description of the components that play roles in this standard.

— Clause 5 describes the cryptographic modes used by the cryptographic unit.

— Clause 6 describes cryptographic key management and initialization vector requirements.

— Annex A (informative) lists bibliographic references that are useful when implementing this standard.

- Annex B (informative) discusses several security issues that an implementer and user should understand.
- Annex C (informative) provides a summary of documentation requirements.
- Annex D (informative) provides several test vectors useful in verifying a cryptographic unit.

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.

IEEE Std 1619™, IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices.[1, 2]

McGrew, David and John Viega, The Galois/Counter Mode of Operation (GCM), May 31, 2005.[3]

NIST FIPS 180-2, Federal Information Processing Standard (FIPS) 180-2 (August 1, 2002), Announcing the Secure Hash Standard (SHA).[4]

NIST FIPS 197, Federal Information Processing Standard (FIPS) 197 (November 26, 2001), Announcing the Advanced Encryption Standard (AES).

NIST FIPS 198, Federal Information Processing Standard (FIPS) 198 (March 2002 updated April 8, 2002), The Keyed-Hash Message Authentication Code (HMAC).

NIST Special Publication 800-38A (NIST SP 800-38A), Recommendation for Block Cipher Modes of Operation—Methods and Techniques.

NIST Special Publication 800-38C (NIST SP 800-38C), Recommendation for Block Cipher Modes of Operation: The CCM mode for authentication and confidentiality.

3. Keywords, definitions, acronyms, and abbreviations

3.1 Keywords

For the purposes of this standard, the following terms are keywords.

can: A keyword indicating a capability (*can* equals *is able to*).

required: *See* **shall**.

may: A keyword indicating a course of action permissible within the limits of this standard (*may* equals *is permitted to*).

[1] IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

[2] The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[3] Available from the World Wide Web site http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf or http://siswg.org/docs/gcm_spec.pdf

[4] Available from the NIST World Wide Web site http://csrc.nist.gov/

must: A keyword used only to describe an unavoidable situation that does not constitute a requirement for compliance to this standard.

shall: A keyword indicating a mandatory requirement strictly to be followed in order to conform to this standard and from which no deviation is permitted.(*shall* equals *is required to*).

shall not: A phrase indicating an absolute prohibition of this standard.

should: A keyword indicating that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (s*hould* equals *is recommended to*).

3.2 Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards, Seventh Edition,* [B6][5] should be referenced for terms not defined in this clause.

3.2.1 additional authenticated data (AAD): Information passed into an authenticated encryption routine that is authenticated but not encrypted.

3.2.2 advanced encryption standard (AES): The block cipher defined by NIST FIPS 197. *See also*: **block cipher**.

3.2.3 block cipher: A cryptographic primitive that uses a cipher key to create a pseudo-random permutation of a fixed-size bit string. *See also*: **cipher key; plaintext; ciphertext.**

3.2.4 cipher block chaining (CBC): A cryptographic mode of operation in which the ciphertext output from each cipher block feeds into the following cipher block (see NIST SP 800-38A).

3.2.5 cipher block chaining initialization vector (CBC-IV): The IV input for the CBC modes, according to NIST SP 800-38A. *See also*: **CBC-HMAC; initialization vector**.

NOTE—See 5.4.[6]

3.2.6 cipher block chaining with keyed-hash message authentication code(CBC-HMAC): A family of cryptographic modes that uses the cipher block chaining (CBC) mode (see NIST SP 800-38A) for confidentiality and a key-hash message authentication code (HMAC) for integrity (see NIST FIPS 198).

3.2.7 cipher block chaining with keyed-hash message authentication code using secure hash algorithm (CBC-HMAC-SHA): A family of cipher blocking chaining (CBC) modes (see NIST SP 800-38A) with a keyed-hash message authentication code (see NIST FIPS 198) from a member of the secure hash algorithm family (see NIST FIPS 180-2).

NOTE—See 5.4.

3.2.8 cipher key: A bit string that controls the pseudo-random permutation of a encryption or decryption routine. *See also:* **cryptographic key; block cipher.**

3.2.9 ciphertext record: The result of encrypting a same-length plaintext record using a cryptographic mode of operation. *See also*: **ciphertext; cryptographic mode of operation; plaintext record**.

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

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

3.2.10 collision: An event where two independent variables have the same value in a particular context. *See also:* **initialization vector; plaintext; ciphertext.**

3.2.11 counter mode (CTR): A cryptographic mode of operation defined by NIST SP 800-38A in which the ciphertext is the bitwise exclusive-OR of the plaintext with an encrypted counter.

3.2.12 counter with cipher block chaining message authentication code (CCM): A cryptographic mode of operation that provides confidentiality with counter mode and integrity with a message authentication code that uses cipher block chaining (see NIST SP 800-38C).

NOTE—See 5.2.

3.2.13 cryptographically-sound random-bit generator (RBG): A device or algorithm that outputs a sequence of binary bits that appears to be statistically independent and unbiased. In particular, a RBG generates numbers that are highly unpredictable, and knowledge of any particular output from a RBG does not reveal any information about other data generated by the RBG.

NOTE—See 6.1.

3.2.14 cryptographic hash function: A hash function that generates a hash value from an input and that has the following properties: 1) it is computationally difficult to compute the inverse (i.e., compute the input from the hash value); 2) it is computationally difficult to find two different inputs that have the same hash value; 3) it is computationally difficult to find an input whose hash value is a particular value.

3.2.15 cryptographic key: A bit string used as an input into cryptographic primitives. *See also*: **block cipher; cipher key.**

3.2.16 cryptographic mode: *See* **cryptographic mode of operation**.

3.2.17 cryptographic mode of operation: An algorithm that includes a block cipher used in a particular configuration and uses a cipher key to convert plaintext into ciphertext and vice versa. *See also*: **block cipher; cipher key; plaintext; ciphertext**.

3.2.18 cryptographic unit: Any set of software, firmware, or hardware that can perform a cryptographic operation.

3.2.19 decryption: The act of producing plaintext from ciphertext. *Contrast:* **encryption.** *See also*: **plaintext; ciphertext; cryptographic key; cryptographic mode of operation**.

3.2.20 decryption routine: An instantiation of a cryptographic mode of operation that converts ciphertext into plaintext.

3.2.21 encrypted record: A collection of fields that includes the output of an encryption operation or authentication operation (e.g., ciphertext, message authentication code), and optionally contains other information needed for a subsequent decryption operation (e.g., additional authenticated data, initialization vector). *See also*: **ciphertext; initialization vector; message authentication code; additional authenticated data**.

3.2.22 encryption: The act of producing ciphertext from plaintext. *Contrast:* **decryption.** *See also*: **plaintext; ciphertext; cryptographic key; cryptographic mode of operation**.

3.2.23 encryption routine: An instantiation of a cryptographic mode of operation that converts plaintext into ciphertext.

3.2.24 encryption session: An interval in which a cryptographic unit generates encrypted records using a set of self-consistent variables, such as unique initialization vectors. *See also*: **encryption; initialization vector; cryptographic unit**.

NOTE—See 6.5.3.

3.2.25 Galois/Counter Mode (GCM): A cryptographic mode of operation that provides confidentiality through counter mode encryption and integrity through a message authentication code that uses Galois field arithmetic. (See McGrew and Viega, The Galois/Counter Mode of Operation.)

NOTE—See 5.3.

3.2.26 host record: A string of plaintext passed to the cryptographic unit from the host. *See also*: **plaintext record; cryptographic unit; host**.

3.2.27 initialization vector (IV): An input into an encryption or decryption algorithm that needs not be secret, but has a high probability of being unique when used with a particular cipher key. *See also*: **encryption; decryption; encryption session; cipher key**.

NOTE—See 6.5.

3.2.28 key encrypting key (KEK): A cryptographic key used only for encrypting or decrypting other cryptographic keys. *See also*: **cryptographic key**; **encryption**.

3.2.29 keyed-hash message authentication code (HMAC): A message authentication code defined by NIST FIPS 198 that includes a secret hash key.

3.2.30 key manager: Any device or person that controls the creation, archiving, and destruction of cryptographic keys. *See also*: **cryptographic key**.

NOTE—See 4.2.3.

3.2.31 message authentication code (MAC): A cryptographic checksum that is used to detect intentional modifications and errors in an encrypted record, and cannot be efficiently forged without knowledge of the cryptographic key used in the MAC algorithm. *See also*: **encrypted record**.

3.2.32 nonce: A bit string that has a low probability of matching any other nonce in a particular context. *See also:* **initialization vector**.

3.2.33 plaintext: Information that has not been obscured through a cryptographic transformation. *Contrast:* **ciphertext.**

3.2.34 plaintext record: A string of plaintext passed to an encryption routine to produce a same-length ciphertext record. *See also*: **plaintext; cryptographic unit; encryption; encrypted record; host record**.

3.2.35 policies: In cryptography, a set of rules that defines aspects of the management of a cryptographic system (e.g., encryption, decryption, or bypass rules).

3.2.36 random bit generator (RBG): *See* **cryptographically-sound random-bit generator.**

3.2.37 random nonce: A nonce that completely consists of the output from a random bit generator. *See also*: **nonce**; **cryptographically-sound random-bit generator**.

3.2.38 Secure Hash Algorithm (SHA): A family of cryptographic hash functions defined by NIST FIPS 180-2. *See also* **cryptographic hash function.**

3.2.39 self-contained group: A set of cryptographic units that generate and use IVs in a consistent manner.

3.2.40 Xor-encrypt-xor with tweak and ciphertext stealing (XTS): The cryptographic mode of operation described in IEEE Std 1619.

NOTE—See 6.6.

3.3 Acronyms and abbreviations

AAD	additional authenticated data
AES	advanced encryption standard
CBC	cipher block chaining
CCM	counter with cipher block chaining message authentication code
CTR	counter mode
FIPS	Federal Information Processing Standards
GCM	Galois/Counter Mode
HMAC	keyed-hash message authentication code
IEEE	Institute of Electrical and Electronics Engineers
IV	initialization vector
KEK	key encrypting key
MAC	message authentication code
NIST	National Institute of Standards and Technology
RBG	(cryptographically-sound) random-bit generator
SHA	Secure Hash Algorithm
SP	(NIST) special publication
XOR	exclusive OR
XTS	xor-encrypt-xor with tweak and ciphertext stealing

3.4 Mathematical conventions

This standard uses decimal, binary, and hexadecimal numbers. For clarity, decimal numbers generally represent counts, and binary or hexadecimal numbers describe bit patterns or raw binary data.

Binary numbers are represented by a string of one or more binary digits, followed by the subscript 2. For example, the decimal number 26 is represented as 00011010_2 in binary.

4. General concepts

4.1 Introduction

This standard describes elements of an architecture that is suitable for the cryptographic confidentiality and integrity of stored data. This architecture includes a model of several components within a typical system that securely stores and retrieves information. These components are as follows:

— A *controller* that controls the overall operation of the cryptographic unit and receives status from the cryptographic unit (see 4.2.1)

— A *host* that provides plaintext data, in the form of host records, to the cryptographic unit and receives plaintext data from the cryptographic unit (see 4.2.2)

— A *key manager* that may provide or negotiate *cipher keys* and/or *key encrypting keys* (KEK) to the cryptographic unit, and that should securely maintain the lifecycle of these cryptographic keys (see 4.2.3)

— A *cryptographic unit* that performs data formatting, encryption, and decryption, and that may perform cryptographic key management (see 4.2.4)

— A *storage medium* that provides non-volatile storage of encrypted records and metadata produced by the cryptographic unit (see 4.2.5)

This standard specifies requirements only for the cryptographic unit.

An implementer of this standard shall provide documentation to the end-user about the cryptographic unit. This documentation may be in any form (e.g., electronic, printed on paper) that is easily accessible by the end-user. Documentation shall include all the required text as specified throughout this standard. The documentation provides sufficient information to allow optimal use and detailed security evaluation of the cryptographic unit and its environment. See Annex C for a documentation summary.

Figure 1 shows an example of the interactions among the five components listed above and of subcomponents contained within each component. Multiple components shown in Figure 1 may exist within a single embodiment, and multiple instantiations of the same component or subcomponent may exist within a single system.

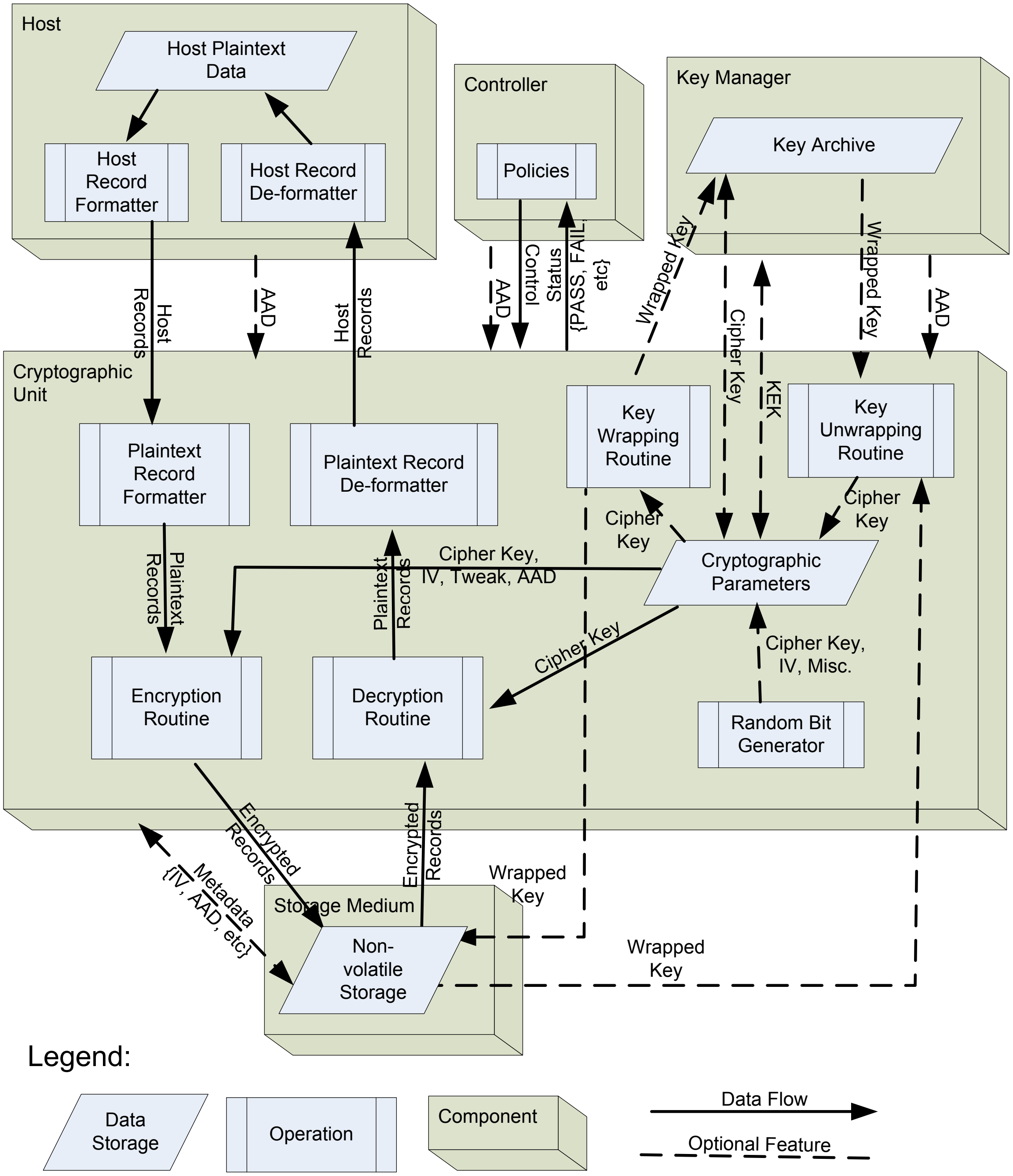

Figure 1—Model showing interactions of components

Subclause 4.2 describes in detail each of the components within Figure 1.

4.2 Components

4.2.1 Controller

The controller is any entity that controls the overall operation of the cryptographic unit. A controller sends commands to the cryptographic unit and processes status from the cryptographic unit, as needed to implement the *policies* defined within the controller. There may be multiple controllers controlling a particular cryptographic unit. A controller may be part of another component such as a host or key manager.

4.2.2 Host

The host provides host records to the cryptographic unit for encryption, and receives host records from the cryptographic unit after decryption. A host record contains plaintext data and may be any size that the cryptographic unit allows.

A typical host includes routines to convert arbitrary host plaintext data into host records and vice versa. Such host records may be variable-length, depending on the capabilities of the cryptographic unit. In Figure 1 these routines are as follows:

- **Host record formatter:** A routine that converts arbitrary host plaintext data into host records for the cryptographic unit
- **Host record de-formatter:** A routine that processes host records from the cryptographic unit into host plaintext data

It is not required for a host to implement these functions. The host needs only to present host records to the cryptographic unit, and accept host records from a cryptographic unit.

Examples:

- If the cryptographic unit is contained in a tape drive, then the host might be a computer running a backup application in which the backup application takes arbitrary host plaintext data in the form of files and consolidates them into backup sets, breaks these backup sets into variable-length blocks, and sends the blocks as host records to the cryptographic unit.
- If the cryptographic unit is contained in a disk drive, then the host might be an operating system that formats files into fixed-size sectors (typically 512 bytes) and uses these sectors as host records when sending data to and receiving data from the cryptographic unit.

4.2.3 Key manager

The key manager is responsible for the life-cycle (e.g., generation, archiving, and destruction) of the cryptographic keys used by the cryptographic unit. Such a cryptographic key may be a cipher key or a key encrypting key (KEK) (see 6.3). The key manager may maintain cryptographic keys within a key archive. Requirements on the key manager and key archive are outside the scope of this standard, but are critical for the security of the complete system (see IEEE P1619.3 [B8]).

4.2.4 Cryptographic unit

A cryptographic unit is any combination of software, firmware, or hardware that is capable of handling plaintext and ciphertext using at least one of the cryptographic modes specified in 5.1.

The cryptographic unit shall contain the following subcomponents:

- Plaintext record formatter (see 4.3) and/or plaintext record de-formatter (see 4.4)
- Encryption routine (see 4.5) and/or decryption routine (see 4.6)
- Cryptographic parameters (see 4.7)

The cryptographic unit may contain the following subcomponents:

- Random bit generator (see 6.1)

- Key wrapping routine (see 6.4)
- Key unwrapping routine (see 6.4)

4.2.5 Storage medium

The storage medium is any device or material capable of non-volatile storage of encrypted records and metadata.

The controller may configure the cryptographic unit to write a particular plaintext record to the storage medium either with encryption or without encryption. The cryptographic unit may mix both encrypted records and plaintext records on the storage medium. The cryptographic unit may write additional information without encryption to the storage medium, assuming that such information does not reveal cryptographic keys or plaintext that was intended to be encrypted. The cryptographic unit shall not write information to the storage medium that compromises the cryptographic confidentiality or integrity of any encrypted information on the storage medium.

4.3 Plaintext record formatter

The plaintext record formatter is a routine that converts host records into plaintext records that pass into the encryption routine. In the simplest case, this routine could simply pass host records directly through as plaintext records. In more complicated systems, this routine could perform compression, padding, or other reversible transforms.

The cryptographic unit receives host records from the host as a basic unit of data for encryption. When performing encryption, the cryptographic unit shall use the plaintext record formatter to format the host records into plaintext records.

To minimize buffering requirements and latency, the cryptographic unit may define a maximum size for the plaintext records that is smaller than the maximum host record size allowed by the cryptographic unit. The plaintext record formatter may split the host record into multiple plaintext records with optional padding or reformatting.

The cryptographic unit may apply padding or perform reversible transforms (such as compression) to the data within the host records to form the plaintext records.

If a host record is formed from two or more plaintext records, then the cryptographic unit shall include sufficient information within the AAD, IV, or plaintext record to allow the plaintext record de-formatter (see 4.4) to unambiguously reconstruct each of the original host records or detect malicious tampering. To help fulfill this requirement, the cryptographic unit should use ordering verification to detect tampering or reordering of the encrypted records (see 4.6.3).

Documentation shall describe how the plaintext record formatter generates plaintext records from host records.

4.4 Plaintext record de-formatter

The plaintext record de-formatter is a routine that converts plaintext records received from the decryption routine into host records that the cryptographic unit passes to the host.

The plaintext record de-formatter shall only use information that the decryption routine is able to cryptographically verify using a message authentication code (MAC).

If the plaintext record contains padding or reversible transforms, then the plaintext record de-formatter shall verify the correctness of these formats. If the format is incorrect, then the cryptographic unit shall send the special signal *FAIL* to the host and/or controller and should not return any host records.

Documentation shall describe how the plaintext record de-formatter generates host records from plaintext records.

4.5 Encryption routine

4.5.1 Overview

The encryption routine takes formatted plaintext records as input and produces encrypted records as output. The following subclauses describe the characteristics of an encryption routine that are common across all cryptographic modes.

4.5.2 Inputs

The encryption routine requires the following inputs (see 5.1 for limits):

a) A secret cipher key
b) An initialization vector (IV)
c) Length of the IV
d) Plaintext record
e) Length of the plaintext record
f) Additional authenticated data (AAD)
g) Length of the AAD

4.5.3 Outputs

The encryption routine produces an encrypted record that contains the following:

a) A ciphertext record;
b) A message authentication code (MAC);
c) Optionally the IV or enough information to reconstruct the IV; and
d) Optionally the AAD or enough information to reconstruct the AAD.

The ciphertext record shall have the same length as the plaintext record. An encrypted record may additionally contain the IV and AAD. If an encrypted record does not contain both the IV and AAD, then there shall be sufficient information on the storage medium or in the cryptographic unit to allow reconstruction of the complete IV and AAD. The IV and AAD may contain any other information that does not compromise the cryptographic confidentiality and integrity of the encrypted record (e.g., information to support ordering verification as given in 4.6.3).

The cryptographic unit shall write the encrypted record to the storage medium.

When performing encryption, the cryptographic unit shall not write the plaintext or cipher key to the storage medium unencrypted. The cryptographic unit may write a cryptographically wrapped version of the cipher key to the storage medium (see 6.4).

For encryption, the cipher key shall be associated with a single cryptographic mode, and the cryptographic unit shall not use this cipher key in any other cryptographic mode. The key manager should associate each cipher key with a single cryptographic mode.

4.6 Decryption routine

4.6.1 Overview

The decryption routine uses a cipher key to convert encrypted records from the storage medium into plaintext records for the plaintext record de-formatter.

The following subclauses describe the requirements for decryption that are common among all the cryptographic modes specified in this standard.

4.6.2 Decryption inputs

The decryption routine requires the following inputs:

a) A secret cipher key
b) Initialization vector (IV)
c) Length of the IV
d) Ciphertext record
e) Length of the ciphertext record
f) Additional authenticated data (AAD)
g) Length of the AAD
h) A message authentication code (MAC) with length determined by the cryptographic mode

During decryption, the cryptographic unit shall always validate the MAC. The cryptographic unit should validate the MAC before sending any plaintext to the host. Best practices recommend validating the MAC before returning plaintext (see B.4 for a discussion on the security concerns of returning plaintext before validating the MAC). Documentation shall disclose whether the cryptographic unit validates the MAC before returning any plaintext.

If the cryptographic unit validates the MAC before returning plaintext, then it shall not return plaintext to the host if the MAC validation fails. If the MAC validation fails, then the cryptographic unit shall return the special signal *FAIL* to the host and/or controller.

If the cryptographic unit returns plaintext to the host before validating the MAC, then the cryptographic unit shall subsequently validate the MAC. If this MAC validation fails, then the cryptographic unit shall return the special signal *FAIL* to the host and/or controller. If this MAC validation passes, then the cryptographic unit shall return the special signal *PASS* to the host and/or controller.

If the cryptographic unit is capable of returning plaintext before validating the MAC, then the host should not act on any plaintext from the cryptographic unit until receiving a complete host record and the special signal *PASS*.

Documentation shall define the special signal *FAIL* and describe how the host and/or controller receive such a signal. The special signal *FAIL* should identify the host records that failed the MAC validation.

If the cryptographic unit is capable of returning plaintext before validating the MAC, then documentation shall define the special signal *PASS*, describe how the host and/or controller receive such a signal, and define limits for the number of host records and bytes of plaintext that the cryptographic may return before checking the MAC.

4.6.3 Ordering verification

During decryption, a cryptographic unit should performing ordering verification by checking that each IV or AAD is consistent with the preceding IV or AAD based on the documented mechanism used for creating the IV or AAD (see B.3).

If a cryptographic unit is performing ordering verification and detects an inconsistent IV or AAD, then the cryptographic unit shall return the special signal *FAIL* to the host and/or controller.

If a cryptographic unit supports ordering verification, then documentation shall specify the methods for enabling or disabling this functionality, and shall specify how the cryptographic unit notifies the host and/or controller of inconsistent IV or AAD ordering, and how to recover, if possible.

4.6.4 Verification-only mode

The cryptographic unit may support a verification-only mode, where it only validates the MAC and returns a *PASS* or *FAIL* signal to the host and/or controller, but does not return any host records.

4.7 Cryptographic parameters

A cryptographic parameter is a value that affects the cryptographic confidentiality or integrity of encrypted information.

The cryptographic unit shall protect the following cryptographic parameters from unauthorized modification while stored in the cryptographic unit, but may permit disclosure:

- Additional authenticated data (AAD)
- Initialization vector (IV)
- Any asymmetric public key [e.g., asymmetric public key encrypting key (KEK)]

Additionally, the cryptographic unit shall protect the following cryptographic parameters from both unauthorized modification and disclosure:

- Cipher keys
- Seed keys for random bit generators
- Any asymmetric private key (e.g., asymmetric private KEK)
- Any symmetric KEK

The cryptographic unit may disclose cryptographic parameters to authorized entities if such disclosure uses cryptographic methods or uses a physically secure connection.

Documentation shall describe all cryptographic parameters used by the cryptographic unit.

5. Cryptographic modes

5.1 Overview

This clause describes the cryptographic modes of operation (i.e., cryptographic modes) allowed by this standard when the cryptographic unit is operating in a compliant mode. The cryptographic unit shall support at least one of the cryptographic modes shown in Table 1.

Table 1—Cryptographic modes

Family	Fully qualified name	Description	Ref.
CCM	CCM-128-AES-256	Counter with 128-bit cipher block chaining MAC	5.2
GCM	GCM-128-AES-256	Galois/Counter Mode with 128-bit MAC	5.3
CBC-HMAC	CBC-AES-256-HMAC-SHA-1	Cipher block chaining with 160-bit HMAC	5.4
	CBC-AES-256-HMAC-SHA-256	Cipher block chaining with 256-bit HMAC	5.4
	CBC-AES-256-HMAC-SHA-512	Cipher block chaining with 512-bit HMAC	5.4
XTS-HMAC	XTS-AES-256-HMAC-SHA-512	Xor-encrypt-xor with tweak and ciphertext stealing, with 512-bit HMAC	5.5

When describing the encryption and decryption routines independently, this standard uses the suffix “-ENC” to denote the encryption routine, and “'DEC” to denote the decryption routine. For example, “CCM-128-AES-256-ENC” refers to the encryption routine implementing the CCM-128-AES-256 cryptographic mode.

The cryptographic unit shall operate these cryptographic modes within the parameter limits given in Table 2.

Table 2—Parameter limits for encryption modes

Cryptographic Mode	**Parameter limits, in bytes**					
	Cipher key	**IV**	**AAD**	**Plaintext record**	**Maximum total plaintext**[b]	**MAC**
CCM-128-AES-256	32	12	0 to $2^{64} - 1$	0 to $2^{24} - 1$	$2^{64} - 1$	16
GCM-128-AES-256	32	12, or 16 to $2^{61} - 1$	0 to $2^{61} - 1$	0 to $2^{36} - 32$	$2^{68} - 16$	16
CBC-AES-256-HMAC-SHA-1	52[a]	16	0 to $2^{64} - 4$ in multiples of 4	0 to $2^{64} - 16$ in multiples of 16	$2^{64} - 1$	20
CBC-AES-256-HMAC-SHA-256	64[a]	16			$2^{64} - 1$	32
CBC-AES-256-HMAC-SHA-512	96[a]	16			$2^{64} - 1$	64
XTS-AES-256-HMAC-SHA-512	128[a]	16	0 to $2^{125} - 128$	0, or 16 to $2^{68} - 1$	$2^{68} - 1$	64

[a] Includes both AES key and MAC key
[b] Applies to all data encrypted during the lifetime of a particular cipher key

All lengths shall be an integer number of bytes (i.e., multiples of 8 bits).

A cryptographic unit may impose parameter limits that are more restrictive than those in Table 2. Documentation shall specify the parameter limits for the cryptographic unit, if different from those in Table 2.

5.2 Counter with cipher block chaining-message authentication code (CCM)

A cryptographic unit that supports the CCM-128-AES-256 cryptographic mode shall use the algorithm specified by NIST Special Publication 800-38C (NIST SP 800-38C) with the following specifications:

a) The block cipher algorithm shall be AES with a 256-bit (32-byte) cipher key (see NIST FIPS 197).

b) The counter generation function shall be as specified in Appendix A of NIST SP 800-38C.

c) The formatting function shall be as specified in Appendix A of NIST SP 800-38C.

d) The MAC length (Tlen) shall be 128 bits (16 bytes).

e) The IV length shall be 96 bits (12 bytes). The IV input to the encryption procedure corresponds to the nonce *N* required by CCM (see NIST SP 800-38C).

f) The cryptographic unit may return plaintext to the host before validating the MAC, as described in 4.6.2.

g) The IV computation shall follow requirements from 6.5.

NOTE—The IV used for the CBC-MAC computation of CCM does not correspond to the CBC-IV used in CBC-HMAC (see 5.4), even though the names are similar. The CBC-MAC portion of CCM uses a “CBC-IV” of all zeros, as compared to CBC-HMAC, which uses a unique CBC-IV for each invocation.

The data length shall be represented using 24 bits (3 bytes), which is parameterized by setting t = 16, and q = 3 (see A.2.1 within NIST SP 800-38C). Table 3 shows the format of block B_0.

Table 3—Formatting of B_0

Byte Number:	0	1...12	13...15
Value:	$0\boldsymbol{y}111010_2$	Initialization Vector (IV)	Byte-length of plaintext

In Table 3, the variable ***y*** shall equal a binary '0' if the AAD length is zero and a binary '1' if the AAD length is non-zero. For example, the first byte of B_0 has the binary value 00111010_2 if there is no AAD and 01111010_2 if there is AAD.

The *Flags* field within the *counter blocks* shall contain the binary value 00000010_2 (see A.3 within NIST SP 800-38C). All other parameters shall be as specified in Appendix A of NIST SP 800-38C.

NOTE—NIST SP 800-38C does not allow any plaintext to be returned if the MAC validation fails. This standard allows an exception to this case as described in 4.6.2.

5.3 Galois/Counter Mode (GCM)

A cryptographic unit that supports the GCM-128-AES-256 cryptographic mode shall use the GCM algorithm specified in McGrew and Viega's *The Galois/Counter Mode of Operation,* with the following parameters:

a) The block cipher algorithm shall be AES with a 256-bit (32-byte) cipher key (see NIST FIPS 197).

b) The MAC length shall be 128 bits (16 bytes). The MAC shall be used as the Tag defined in the GCM algorithm.

c) The IV computation shall follow requirements from 6.5.

d) The cryptographic unit may return plaintext to the host before validating the MAC, as described in 4.6.2.

e) The length of the IV shall be either 12 bytes, or between 16 bytes and $2^{61} – 1$, inclusive.

NOTE 1—The document entitled "The Galois/Counter Mode of Operation" by Mcgrew and Viega does not allow any plaintext to be returned if the MAC validation fails. This standard makes an exception for this case as described in 4.6.2.

NOTE 2—Using an IV with more than 128 bits (16 bytes) does not add more security because a long IV is distilled back to 16 bytes before use.

5.4 Cipher block chaining with keyed-hash message authentication code (CBC-HMAC)

A cryptographic unit that supports a cryptographic mode within the CBC-HMAC family shall use the CBC mode specified by NIST Special Publication 800-38A (NIST SP 800-38A) and HMAC as specified in NIST FIPS 198, with the following specifications:

a) The block cipher algorithm shall be AES with a 256-bit (32-byte) AES key (see NIST FIPS 197).

b) The HMAC shall use one of the following hashing functions (see NIST FIPS 180-2):

   1) SHA-1;

   2) SHA-256; or

   3) SHA-512.

c) The MAC length (i.e., Tlen) shall match the output length of the underlying hash function [e.g., 160 bits (20 bytes) for SHA-1, 256 bits (32 bytes) for SHA-256, or 512 bits (64 bytes) for SHA-512].

d) The MAC key length shall be equal to the MAC length (i.e., Tlen).

e) The plaintext record length shall be a multiple of 16 bytes (see 4.3 for a discussion on padding).

f) The AAD length shall be a multiple of 4 bytes.

g) The cryptographic unit shall compute IVs, called CBC-IVs in the case of CBC-HMAC, according to one of the following methods:

   1) Set the CBC-IV to a random IV (see 6.5.2); or

   2) Set the CBC-IV to the result of encrypting a nonce IV (see 6.5.3) with the AES block cipher, using the AES key (see NIST SP 800-38A, Appendix C).

h) The CBC-IV length shall be 128 bits (16 bytes).

i) The cryptographic unit may return plaintext to the host before validating the MAC (see 4.6.2).

NOTE 1—Even though the plaintext record is required to be a multiple of 16 bytes, the host record may be any size if the plaintext record formatter of the cryptographic unit provides padding to produce plaintext records that are a multiple of 16 bytes.

For CBC-HMAC, the cryptographic unit shall compute HMAC over the concatenation of the AAD, CBC-IV, and ciphertext record. If the AAD has variable-length, then there shall be sufficient information within the AAD to allow the decryption routine to unambiguously determine where the AAD ends and the CBC-IV starts.

If the cryptographic unit supports CBC-HMAC, then documentation shall describe the format of the AAD, and the method used to determine where the AAD ends and the CBC-IV starts. During decryption, the cryptographic unit shall use this method to determine where the AAD ends and shall send the special signal *FAIL* to the host and/or controller if the AAD does not adhere to the documented format.

NOTE 2—It is possible to fulfill the previous requirement by including the length of the AAD within a fixed-length field at the beginning of the AAD.

For CBC-HMAC, the cipher key length shall be 416 bits when using CBC-AES-256-HMAC-SHA-1, 512 bits when using CBC-AES-256-HMAC-SHA-256, and 768 bits when using CBC-AES-256-HMAC-SHA-512. The cryptographic unit shall use the first 256-bits of the cipher key as the AES key in the encryption and decryption routines. The cryptographic unit shall use the remaining bits of the cipher key as the HMAC key in the MAC generation and verification routines.

Figure 2 shows the CBC-AES-256-HMAC-SHA encryption routine.

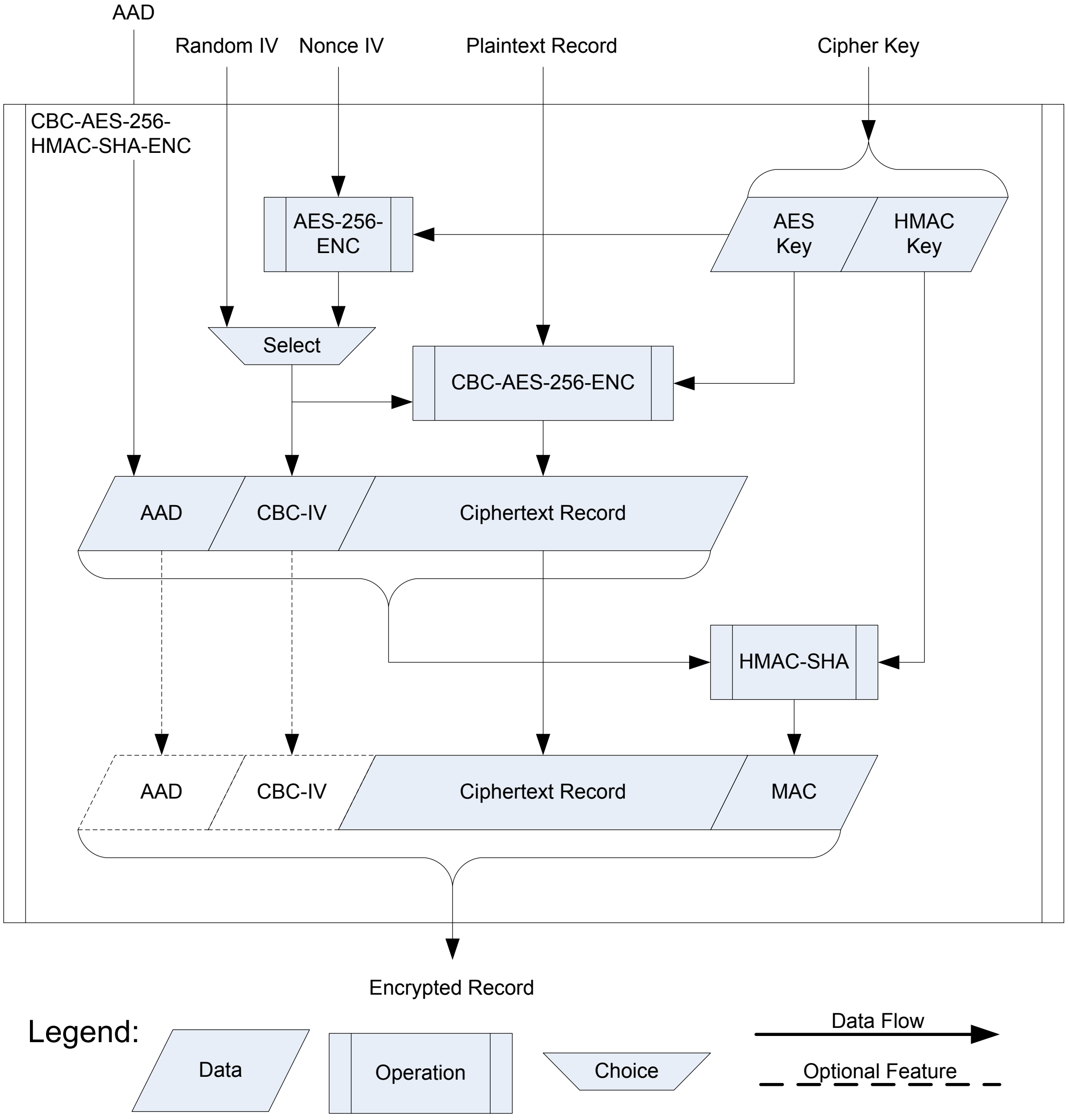

Figure 2—Depiction of a CBC-AES-256-HMAC-SHA encryption routine

In Figure 2, the dotted lines around the "AAD" and "CBC-IV" boxes indicate that it is optional to include these fields within an encrypted record if there is enough information elsewhere to reconstruct the AAD and CBC-IV for the decryption routine.

NOTE—Even though the 'Select' box in Figure 2 shows two possible inputs, a particular implementation that supports CBC-HMAC is only required to support one of these choices.

5.5 Xor-encrypt-xor with tweakable block-cipher with keyed-hash message authentication code (XTS-HMAC)

A cryptographic unit that implements a cryptographic mode within the XTS-HMAC family shall use the XTS-AES-256 procedure as specified in IEEE Std 1619 for confidentiality, and HMAC-SHA-512 as specified by NIST FIPS 198 and NIST FIPS 180-2 to generate the MAC, with the following specifications:

a) The cipher key length shall be 1024 bits (128 bytes), consisting of the concatenation of the following parts, in order:

   1) An AES key that is 512 bits (64 bytes) in length, used as input into the XTS-AES-256 procedure (see IEEE Std 1619)

   2) An HMAC key that is 512 bits (64 bytes) in length, used as input into the HMAC-SHA-512 procedure

b) The cryptographic unit shall compute IVs according to 6.5. The IV is used as the tweak specified in IEEE Std 1619.

c) The IV length shall be 128 bits (16 bytes).

d) The resulting MAC shall be 512 bits (64 bytes) in length.

For XTS-HMAC, the cryptographic unit shall compute the HMAC over the concatenation of the AAD, tweak, and ciphertext record. If the AAD has variable-length, then there shall be sufficient information within the AAD to allow the decryption routine to unambiguously determine where the AAD ends and the tweak starts.

If the cryptographic unit supports XTS-HMAC, then documentation shall describe the format of the AAD, and the method used to determine where the AAD ends and the tweak starts. During decryption, the cryptographic unit shall use this method to determine where the AAD ends and shall send the special signal *FAIL* to the host and/or controller if the AAD does not adhere to the documented format.

NOTE 1—It is possible to fulfill the previous requirement by including the length of the AAD within a fixed-length field at the beginning of the AAD.

NOTE 2—Even though the plaintext record is required to be at least 16 bytes long, the host record may be smaller if the plaintext formatter of the cryptographic unit provides padding.

Figure 3 shows the XTS-AES-256 encryption routine.

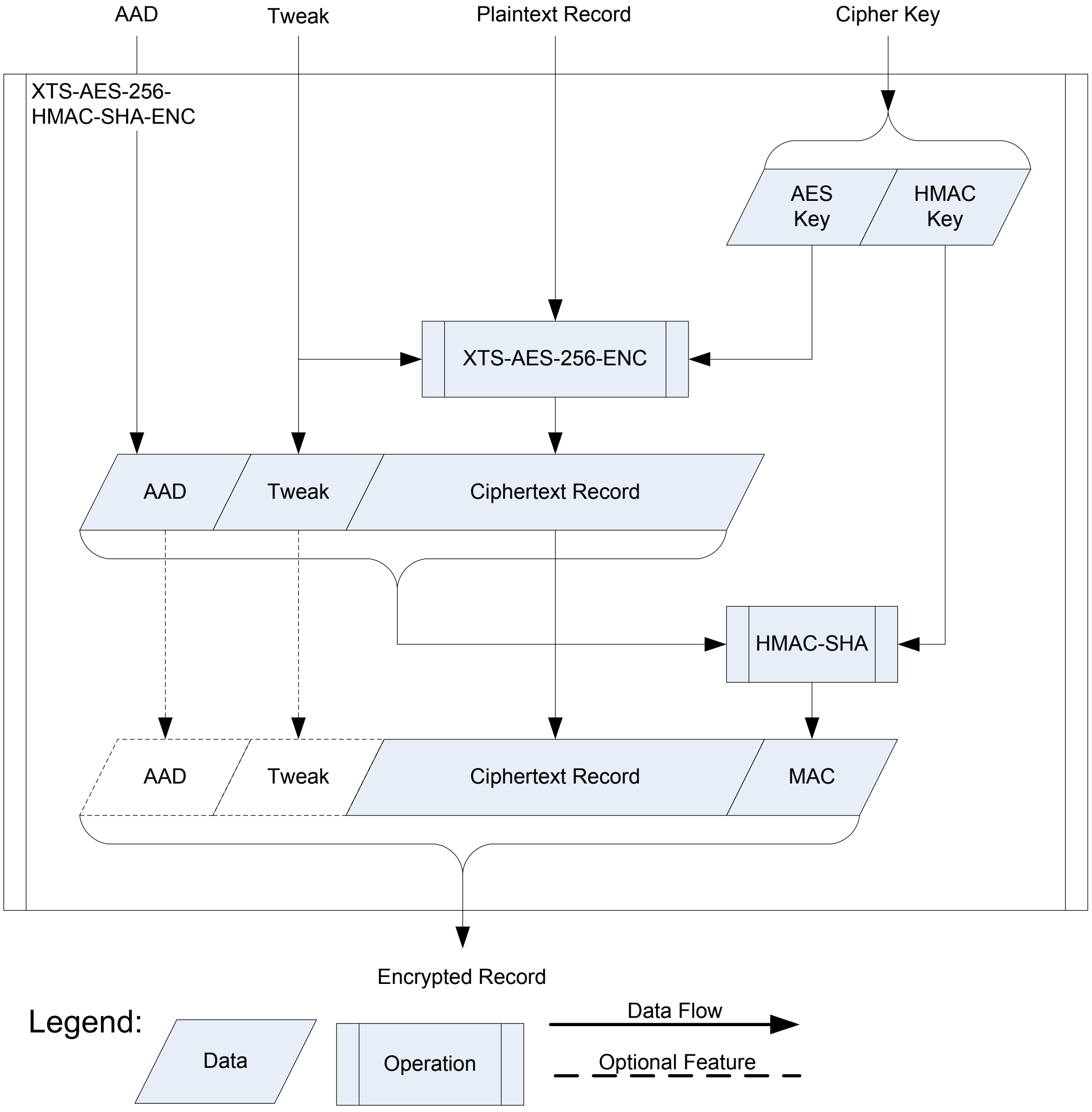

Figure 3—Depiction of an XTS-AES-256 encryption routine

In Figure 3, the dotted lines around the “AAD” and “Tweak” boxes indicate that it is optional to include these fields within an encrypted record if there is enough information elsewhere to reconstruct the AAD and Tweak for the decryption routine.

6. Cryptographic key management and initialization vector requirements

6.1 Random bit generator

If a cryptographic unit needs random data, then the cryptographic unit shall use a cryptographically-sound random-bit generator (RBG) to generate this random data. In particular, the RBG shall be designed such that it is not computationally feasible to predict subsequent outputs from the RBG based on knowledge of previous outputs from the RBG or unencrypted information passed into the cryptographic unit.

A cryptographic unit should implement an RBG that is compatible with NIST FIPS 140-2 [B15] or comparable standards. For implementation guidance describing suitable RBGs, see ANSI X9.31:1998 [B1], ISO/IEC 18031 [B11], Keller [B13], and NIST SP 800-90 [B19].

Documentation shall describe the RBG employed by the cryptographic unit, including algorithms and descriptions of sources of randomness.

6.2 Cryptographic key entry and export

A cryptographic unit should receive cryptographic keys from the key manager using a secure method (e.g., physically secure interface, cryptographically protected communications).

A cryptographic unit shall not make plaintext cryptographic keys available, except to authorized entities that use a physically secure port. A cryptographic unit may make encrypted (i.e., wrapped) cryptographic keys available externally.

The key manager should refrain from entering the same cipher key into both a compliant cryptographic unit and a non-compliant cryptographic unit for purposes of encryption. In this situation, it is possible for the non-compliant cryptographic unit to compromise the security of the data (e.g., the non-compliant cryptographic unit may use the same sequence of IVs as the compliant cryptographic unit).

If the cryptographic unit supports cryptographic key entry or export, then documentation shall specify the supported cryptographic key entry and export methods.

6.3 Handling the cipher key

The cryptographic unit shall use one or more of the following methods to create the cipher key:

a) Generate a new cipher key using only the output from an RBG and perform the following actions:
   1) Create a wrapped cipher key using a KEK from the key manager; and then
   2) Archive the wrapped cipher key using one or more of the following actions:
      i) Store the wrapped cipher key on the storage medium; and/or
      ii) Export the wrapped cipher key to the key manager.

b) Use a cipher key from the key manager and include random information within the IV, as specified in 6.5.2 or 6.5.3.3.

c) Use a cipher key from the key manager and use unique IVs within a self-contained group as specified in 6.6.

The cryptographic unit may use key wrapping, as specified in 6.4, in conjunction with any of the items in the previous list.

NOTE—The controller needs to be especially careful when configuring the cryptographic unit to use a cipher key from the key manager. In this case, it is important for the key manager to frequently generate new cipher keys because the risk of information leakage increases with the square of the amount of plaintext encrypted under the same cipher key (see B.7 and B.8).

6.4 Cryptographic key wrapping on the storage medium

Key wrapping is the process of using a *key encrypting key* (KEK) to encrypt another cryptographic key (e.g., cipher key) using a key wrapping routine. *Key unwrapping* is the process of using a KEK to decrypt a previously wrapped cryptographic key. If the same KEK is used for both wrapping and unwrapping, then it is a *symmetric KEK*. If a different KEK is used for wrapping and unwrapping, then an *asymmetric public KEK* performs the wrapping and an *asymmetric private KEK* performs the unwrapping.

Examples of key wrapping routines are as follows:

- NIST AES key wrap [B14] with a symmetric KEK
- RSAES-OAEP (see RSA PKCS #1 v2.1 [B20]) with an RSA public key as the asymmetric public KEK for encryption and an RSA private key as the asymmetric private KEK for decryption
- ECIES (see IEEE Std 1363a™-2004 [B7]) with an elliptic curve public key as the asymmetric public KEK for encryption and an elliptic curve private key as the asymmetric private KEK for decryption

Support of key wrapping is optional. The cryptographic unit may use any key wrapping routine for protecting the cipher key during import or export, or for archival within the storage medium or key manager. The cryptographic unit should only use key wrapping routines that have undergone peer review within the cryptographic community, such as those listed above.

When unwrapping a wrapped cipher key that was wrapped with an asymmetric public KEK, the key manager should not pass an asymmetric private KEK to the cryptographic unit. Instead, the key manager should retrieve the wrapped cipher key, use its asymmetric private KEK to unwrap it, and then pass the cipher key to the cryptographic unit using a secure method.

If the cryptographic unit supports key wrapping, then documentation shall describe all key wrapping routines that the cryptographic unit supports.

NOTE—The strength of the KEK may affect the strength of the overall solution. See B.2 for a discussion on security concerns of the KEK.

6.5 Initialization vector (IV) requirements

6.5.1 Overview

Encrypting each plaintext record requires a cipher key and an IV, and using the same combination of cipher key and IV to encrypt more than one plaintext record introduces security vulnerabilities (see B.6). To minimize the chances of using the same combination of cipher key and IV to encrypt more than one plaintext record, the cryptographic unit shall generate the IVs according to one of the following methods:

a) **Random IV:** For each encrypted record, the cryptographic unit generates a new IV that consists entirely of the output from an RBG (see 6.5.2).

b) **Nonce IV:** Use encryption sessions, according to 6.5.3.

6.5.2 Using random IVs

A cryptographic unit may generate a random IV as input into each encrypted record. Such a random IV shall entirely consist of the output from an RBG.

6.5.3 Encryption sessions

6.5.3.1 Overview

An encryption session is an interval in which one or more cryptographic units maintain a consistent sequence of IVs for encrypting plaintext records.

A cryptographic unit may maintain multiple independent encryption sessions simultaneously in which each independent encryption session uses a different cipher key and independent IVs.

6.5.3.2 Beginning of a encryption session

Before starting an encryption session, the controller shall configure the cryptographic unit to use a particular method for creating or retrieving the cipher key, according to 6.3.

An encryption session shall begin only after one of the following events:

a) The cryptographic unit receives a cipher key from the key manager.
b) The cryptographic unit generates a new cipher key.

6.5.3.3 Encryption session IV requirements

The cryptographic unit shall encrypt each plaintext record with an IV that is unique within the encryption session. This requirement prevents plaintext leakage within an encryption session (see B.6).

When starting an encryption session, the cryptographic unit shall set the IV to an initial value. If the cryptographic unit uses a cipher key from the key manager (see 6.3), then the initial value for the IV shall contain at least 64 bits that are derived from an RBG. This initial value may continue from the last IV of the previous encryption session if the last IV is part of a consistent sequence that originally started from an initial value containing at least 64 random bits.

Documentation shall describe the format of the IV and the cryptographic unit's mechanism for generating each IV.

6.5.3.4 End of an encryption session

An encryption session shall end after one of the following events:

a) The cryptographic unit receives a command from the controller to end an encryption session. Support of such a command is optional.
b) The cryptographic unit loses the encryption session state, including the cipher key and IV. In this case, the cryptographic unit should notify the host.
c) The cryptographic unit is unable to create an IV that is unique within the encryption session. In this case, the cryptographic unit should send the special signal *FAIL* to the host and/or controller.

If a cryptographic unit is encrypting data in encryption sessions and the encryption session ends, then the cryptographic unit shall not encrypt any more data until another encryption session starts.

If the cryptographic unit supports a command that ends an encryption session, then documentation shall describe this command.

6.6 Creating unique IVs within a self-contained group

This subclause defines requirements for creating unique IVs within a self-contained group of cryptographic units. Support of these requirements is mandatory for cryptographic units that support 6.3 c), and optional otherwise.

When creating unique IVs within a self-contained group, the following statements apply:

a) Cryptographic units may share a common cipher key.

b) Compliance to this standard shall only apply to the entire self-contained group, not an individual cryptographic unit within the group.

c) All cryptographic units within the self-contained group shall be configured to coordinate the creation of unique IVs.

d) Cryptographic units shall not share cipher keys with any cryptographic unit that is not a member of the self-contained group unless such a non-member of the self-contained group includes random information within the IV as defined in 6.3 b).

e) Cryptographic units shall only be used with external services that ensure and document compliance with statement d).

f) Documentation shall describe how the system prevents reuse of the same IV between any two cryptographic units within the self-contained group and how the cryptographic units are uniquely identified. Such identification should use cryptographic methods.

Annex A

(informative)

Bibliography

[B1] ANSI X9.31:1998, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), 1998.[7]

[B2] Biham, E., *New Types of Cryptanalytic Attacks Using Related Keys,* Advances in Cryptology—EUROCRYPT'93, Springer-Verlag, 1994, pp. 398–409.

[B3] Canetti, Ran, Shai Halevi, and Michael Steiner. "Mitigating Dictionary Attacks on Password-Protected Local Storage." Advances in Cryptology—CRYPTO '06. LNCS vol. 4117, pages 160–179. Springer-Verlag, *available from the World Wide Web site* <http://eprint.iacr.org/2006/276>, 2006.

[B4] ECRYPT IST-2002-507932 D.SPA.16, ECRYPT Yearly Report on Algorithms and Keysizes (2005), Jan 2006.

[B5] Ferguson, Niels, Authentication weaknesses in GCM, *available from the World Wide Web site* http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf.

[B6] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms,* Seventh Edition, New York, Institute of Electrical and Electronics Engineers, Inc.

[B7] IEEE Std 1363a-2004, IEEE Standard Specifications for Public-Key Cryptography Amendment 1: Additional Techniques.

[B8] IEEE P1619.3 (Draft 1, May 2007), Draft Standard for Key Management Infrastructure for Cryptographic Protection of Stored Data.[8]

[B9] IETF RFC 2898, PKCS #5: Password-Based Cryptography Specification Version 2.0., *available from the World Wide Web site* http://www.ietf.org/rfc/rfc2898.txt, September 2000.

[B10] IETF RFC 3766, Determining Strengths For Public Keys Used For Exchanging Symmetric Keys, *available from the World Wide Web site* http://www.ietf.org/rfc/rfc3766.txt, April 2004.

[B11] ISO/IEC 18031, Information Technology—Security techniques—Random bit generation, November 2005.[9]

[7] ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

[8] This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE.

[9] ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, CO 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

[B12] Joux, Antoine, Authentication Failures in NIST version of GCM, *available from the World Wide Web site* http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf.

[B13] Keller, Sharon S., NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms, January 2005.

[B14] NIST AES Key Wrap Specification, November 2001.

[B15] NIST FIPS 140-2, Federal Information Processing Standard 140-2, Announcing the Standard for Security Requirements for Cryptographic Modules.

[B16] NIST Draft Special Publication 800-38D (June 27, 2007), Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.

[B17] NIST Special Publication 800-57, Recommendation for Key Management—Part 1: General (Revised), May 2006.

[B18] NIST Special Publication 800-63, Electronic Authentication Guideline: Recommendations of the National Institute of Standards and Technology, April 2006.

[B19] NIST Special Publication 800-90, Recommendation for Random Number Generation Using Deterministic Random Bit Generators.

[B20] RSA PKCS #1 v2.1: RSA Cryptography Standard, June 2002.

Annex B

(informative)

Security concerns

B.1 Threat model

This standard is meant to protect stored data in settings where an attacker might have full access to the storage medium: It is assumed that the attacker might be able to read the content of the storage medium and can also write to it, including replacing some of the stored data with arbitrary data of the attacker's choosing. It is also assumed that the attacker might have access to very large amounts of encrypted data. Such a threat model is suitable for situations where the storage medium is not tightly bound to the cryptographic unit. A prime example is tape encryption, where it is expected that encrypted cartridges are routinely accessed separately from the tape drive where they were first written.

Furthermore, this standard is meant to offer some protection even in highly adversarial situations where an attacker can have repeated access to the storage media of a live system, and can monitor or modify the storage as it is repeatedly being written and over-written. However, only limited protection is provided against replay attacks (see B.3).

Beyond watching and modifying ciphertext, the attacker may have some known- or chosen-plaintext capabilities. This means that the attacker may have some a priori knowledge of the plaintext corresponding to ciphertext that is written on the storage medium, and it may even be able to influence the host into writing plaintext records containing text of the attacker's choosing. This is a realistic assumption when the host represents a multi-user system and some of these users are not highly trusted, and can be realistic even in single-user cases (e.g., when the content of a web cache is written to encrypted disk).

On the other hand, this threat model does not cover security of the information in transit (i.e., how the cryptographic unit receives and sends data to be securely stored). It also does not cover most aspects of key management, such as the generation, transfer, and secure storage of the keys (and does not address mis-management of keys such as using a cryptographic transformation to encrypt its own key). Many of these aspects are addressed by other standards. Finally, a variety of physical (side-channel) attacks against the cryptographic unit, such as timing, power, radiation, fault injection, and the design of a secure random-bit generator (RBG) are out of scope.

B.2 Maintaining cryptographic key security

The security of a cryptographic unit depends on high-quality cryptographic keys. Ideally, the cryptographic keys should come from a cryptographically-sound random-bit generator. The user should not use sources that lack randomness, such as passwords, for any cryptographic key. It is relatively easy for an attacker to launch an off-line dictionary attack against passwords. NIST provides guidelines that estimate the amount of randomness within common passwords (see NIST SP 800-63 [B18]). See also Canetti, et al. [B3] and IETF RFC 2898 [B9].

The effective strength of the solution is determined by many factors, including the strength of the cipher key, the wrapping keys, and keys that are used for ensuring communication security, as well as many aspects of secure authorization. See ECRYPT IST-2002-507932 [B4], IETF RFC 3766 [B10], and NIST SP 800-57 [B17] for different estimates of equivalent key sizes between symmetric key and various asymmetric key encryption algorithms.

B.3 Replay attacks

An implementer should keep the encrypted records in proper logical sequence on a particular storage medium. Otherwise, the cryptographic unit is vulnerable to a *replay attack*, where the attacker replaces a record on the media with some other properly-authenticated record (such as a prior version of the current record).

The attack is applicable in environments where the attacker can read and write directly to the storage medium, and can reorder or repeat encrypted records, assuming the cryptographic unit does not validate the ordering. This could be a powerful attack if the adversary has extensive knowledge of the plaintext and wishes to change the contents of a backup set by replacing specific records with other records, all encrypted with the same cipher key.

Many network encryption standards, such as IPsec, use sequence numbers to handle replay attacks. Similarly, a cryptographic unit should keep the records in sequential order. This could be simply accomplished by including the sequential record number within the AAD or IV fields.

Maintaining a sequential record number helps, but does not handle the case of append operations that overwrite previous data. The addition of a write-pass counter helps ensure that the data has both the correct record number and was written in the correct sequence. Otherwise, an attacker could replay records from either a different tape or a previous write pass of the current tape. A validated write-pass count would prevent this attack.

Unfortunately, including the record number and write-pass count within the AAD or IV field would make it hard, if not impossible, to perform a direct copy of the raw encrypted data from one tape to another. Transferring encrypted data would only be possible by performing an entire tape copy.

Ordering verification does not seem to be applicable in environments where the host has random-access to the storage, such as in a hard drive. Protecting against replay attacks in these environments must therefore be done by other means, such as controlling the access to the media when possible, or relying on a higher-level application. This is beyond the scope of this standard.

Some level of operational protection against replay attacks might be provided by not allowing direct read and write (raw) of encrypted records on the storage medium. This increases the difficulty for attackers by depriving them of ready-made tools to use in these attacks, instead forcing them to implement their own tools. It is clear, however, that this operational protection is only effective against casual attackers. Determined, well-financed attackers can always build their own tools.

B.4 Passing plaintext to the host before checking the MAC

This standard makes an allowance for cryptographic units to pass plaintext to the host before checking the message authentication code (MAC). The purpose is to allow implementations to be compliant even if they are unable to store an entire decrypted host record before passing it back to the host. However, such implementations might have difficulty in gaining certifications such as FIPS 140-2 [B15]. Additionally, both NIST SP 800-38C and NIST SP 800-38D [B16] require that the device validate the MAC before returning any plaintext.

Because of this allowance, it is important that the host does not act on any decrypted plaintext before the MAC validation finishes. Data encrypted using CTR (counter) mode (e.g., GCM, CCM) is especially malleable to an attacker because flipped bits in the ciphertext directly flip corresponding bits in the plaintext. It is also easy to modify any CRC embedded within the plaintext because CRC residuals are linear and depend only on the other modified bits within the ciphertext. If the attacker has knowledge of the plaintext, it is easy to make arbitrary changes to compromise the data. CBC is significantly less vulnerable and XTS is not vulnerable to this attack.

The cryptographic unit can provide operational protection against casual attackers by not implementing commands that allow direct (raw) read and write of encrypted records on the storage medium (see B.3).

B.5 Checking for integrity of a cryptographic key

Use of a corrupted cipher key by the cryptographic unit might lead to wasted resources, and might even open an exposure to related-key attacks (see Biham [B2]). For example, when using one of the HMAC modes in this standard (see 5.4 and 5.5), a cipher key in which only some of the first 256 bits are corrupted would cause an encrypted record to pass the MAC-check but still be decrypted to something other than the original plaintext record.

To avoid using corrupted keys, some measures should be taken to verify the integrity of the cipher key in use. One method for ensuring key integrity is using a "key signature." For example, this could be an HMAC computed over the cipher key using some higher-level key. For further guidance, see NIST SP 800-57 [B17].

B.6 Avoiding collisions of initialization vectors

All the modes that are specified in this standard rely on an initialization vector (IV) that is assumed to be non-repeating within the scope of the cipher key in use. In all of them, using the same combination of cipher key and IV to encrypt two different records (referred to as an *IV*-collision) result in some exposure to attacks and leaks information about the plaintext.

This exposure is particularly acute in modes such as CCM and GCM that use counter-mode encryption (i.e., a block cipher turned into a stream cipher). In these modes, re-use of the same IV under the same cipher key poses the same risk as re-use of the key stream in a stream cipher—namely, the exclusive-OR of the two ciphertext records equals to the XOR of the two plaintext records, allowing an attacker to learn information about the plaintext records by observing the ciphertext. Moreover, an IV collision in GCM might in some circumstances reveal information about the authentication key (which is generated internal to the GCM algorithm) to an attacker, thus allowing the attacker to forge authentication tags (see Ferguson [B5] and Joux [B12]).

This exposure is less acute in the other two modes (CBC and XTS), but it exists even there. For example, an IV collision lets the attacker see if two encrypted records are the same, or even if some specific blocks in these records are the same. To maintain the security of the encryption modes in this standard, it is therefore important to takes proactive steps to avoid IV collisions.

When considering the probability of IV collisions, it is important to take into account the possibility that the same cipher key is loaded into cryptographic units from different manufacturers, and that these cryptographic units may use very different strategies for IV collision avoidance. Therefore, it is important that each cryptographic unit guarantees some level of IV collision security, regardless of the behavior of any other cryptographic units that may be given the same cipher key, or there is a guarantee that the key manager and cryptographic unit maintains a consistent state between the cipher key and IV (see 6.5).

Some examples of IV collision avoidance strategies are described in B.7, along with analysis of their effectiveness in various settings.

B.7 Examples of IV collision avoidance strategies

B.7.1 Example 1: Using random IVs

In this example, the cryptographic unit receives a cipher key from the key manager, to be used with some cryptographic mode that employs *n*-bit IVs. With each encrypted record, the cryptographic unit extracts *n* bits from its RBG for use as the record's IV.

To analyze the effectiveness of this strategy, one can observe that any two IVs assume the same value with probability exactly 2^{-n} (since all the IVs are random). When encrypting c records with the same cipher key, there are $c(c- 1)/2$ (i.e., c choose 2) possible collision events, each occurring with probability 2^{-n}. Using Boole's inequality (the union bound), one can derive the upper-bound on the probability of IV collisions given in Equation (1)

$$p \le \frac{c(c-1)}{2^{n+1}} < \frac{c^2}{2^{n+1}} \tag{1}$$

where

p is the probability of any IV collision occuring
c is the number of records encrypted under a particular cipher key
n is the IV length, measured in bits

For example, when using a 128-bit IV and encrypting 2^{40} (about 10^{12}) records under the same cipher key, the probability of an IV collision is bounded by $p < (2^{40})^2 / 2^{128+1} = 2^{-49}$ (about 1 in 5.63×10^{14}). When encrypting the same 2^{40} records with 96-bit IVs, the bound on the IV collision probability from Equation (1) is: $p < (2^{40})^2 / 2^{96+1} = 2^{-17}$ (1 in 131072).

The bound from Equation (1) is rather tight in this case. In fact, a slightly more complicated argument (using the inclusion-exclusion principle and the fact that these collision events are pair-wise independent) implies an almost matching lower-bound on the probability of collisions in this case, given by Equation (2).

$$p \ge \varepsilon(1-\frac{\varepsilon}{2}+2^{-n-1}) \tag{2}$$

where

$$\varepsilon = \frac{c(c-1)}{2^{n+1}}$$

In the example from above with 128-bit IVs and 2^{40} records, this lower-bound on the collision probability is $p > 2^{-49.000000000001314}$ (still about 1 in 5.63×10^{14}), and in the example with 96-bit IVs and 2^{40} records the lower bound is $p > 2^{-17.0000055}$ (1 in 131072.5).

B.7.2 Example 2: Incrementing a random IV

In this example, the cryptographic unit again receives from the key manager a cipher key with some cryptographic mode that employs n-bit IVs. Upon receiving the key from the key manager, the cryptographic unit extracts n bits from its RBG and stores these bits in a state register. With each encrypted record, the cryptographic unit uses the current contents of the state register as the IV and then increments, modulo 2^n, the state register by one.

The effectiveness of this strategy relies to a large extent on the key manager. For example, a key manager that never reloads the same cipher key in two different encryption sessions effectively guarantees that IV-collisions never occur with this strategy. On the other hand, a "worst-case key manager" that always reloads the same cipher key for encryption of every record would cause IV collisions with the same probability as in the random-IV example from B.7.1.

To get a more quantitative answer, one can observe that any two IVs still only assume the same value with probability no more than 2^{-n}: Two IVs either belong to the same sequence, in which case they cannot possibly collide (assuming that no sequence is longer than 2^n records), or belong to two different sequences, in which case they are derived from independent outputs of the RBG and can only collide with probability 2^{-n}. Hence Equation (1) can still be used to obtain an upper-bound on the probability of IV collisions.

In the current example, the expression from Equation (1) is a very conservative upper bound that holds regardless of the behavior of the key manager, and the actual collision probability can sometimes be much smaller. For example, assume that the same cipher key is used in only *S* encryption sessions, and that each encryption session encrypts *R* different records (so the total number of records is $R \times S$). Notice that for any two encryption sessions, the probability of an IV collision between any IVs in these two encryption sessions is $(2R - 1)/2^n < R/2^{n-1}$, since the initial nonces in these two encryption sessions must be within $R - 1$ of each other for any collision to occur. As there are *S* encryption sessions, one can again use Boole's inequality to upper-bound the probability that any two of them give rise to collisions, as shown in Equation (3).

$$p < \left(\frac{S(S-1)}{2}\right)\left(\frac{R}{2^{n-1}}\right) < \frac{S^2 R}{2^n} \tag{3}$$

where

p is the probability of any IV collision occuring
S is the number of encryption sessions under a particular cipher key
R is the number of records that are encrypted in every encryption session
n is the IV length, measured in bits

For example, with $S = 2^{25}$ encryption sessions, each with $R = 2^{15}$ encrypted records (so the total is still 2^{40} encrypted records), we get an upper bound of $p < 2^{-63}$ when using 128-bit IVs and $p < 2^{-31}$ when using 96-bit IVs [compared to the bounds of 2^{-49} and 2^{-17}, respectively, taken from Equation (1)]. The same argument as in B.7.1 can also be used here to show that this bound is quite tight.

B.7.3 Example 3: Randomizing only the key

In this example, the cryptographic unit chooses a fresh cipher key for every encryption, and uses the integer *I* as the IV for the I^{th} record in an encryption session. Clearly, no collisions are possible between IVs in the same encryption session, and therefore the only risk is key-collision, which happens with negligible probability (since the encryption keys are at least 256-bit long).

B.8 How many records to encrypt with one key?

The bounds from the examples in B.7 can be used as guidance for the maximum amount of data to encrypt with a single cipher key. Specifically, given the maximum acceptable probability of an IV-collision, and knowledge of the cryptographic mode and the collision-avoidance strategy used by the cryptographic unit, one may set an upper bound for data to encrypt with a single cipher key.

Table B.1 contains the maximum number of records for a single key for the collision-avoidance strategies in the examples from B.7.1 and B.7.2. For random IVs, the expression from Equation (2) is used (which is heuristically a bit more accurate than the expression in Equation (1) for large values of ε), and for incrementing random IVs, the expression from Equation (3) with $R = 2^{15}$ is used (i.e., assuming that each encryption session is used to encrypt 2^{15} records).

It should be stressed, however, that the bounds in Table B.1 only consider the probability of IV-collisions, and in most settings there are many other considerations that must be taken into account. For example, 5.1 includes some other limits on the amount of data that can be encrypted per cipher key, and XTS and CBC encryption modes entail their own limitations, as discussed in their respective standards (IEEE Std 1619 and NIST SP 800-38A, respectively).

Table B.1—Maximum number of encrypted records per key

Maximum acceptable probability of IV-collisions	**96-bit IVs**		**128-bit IVs**	
	Equation (2)	**Equation (3), $\boldsymbol{R} = \mathbf{2^{15}}$**	**Equation (2)**	**Equation (3), $\boldsymbol{R} = \mathbf{2^{15}}$**
$p = 2^{-40}$ (~1 in 10^{12})	~ 3.80×10^{8}	~ 4.86×10^{10}	~ 2.49×10^{13}	~ 3.18×10^{15}
$p = 2^{-30}$ (~1 in 10^{9})	~ 1.21×10^{10}	~ 1.55×10^{12}	~ 7.96×10^{14}	~ 1.02×10^{17}
$p = 2^{-20}$ (~1 in 10^{6})	~ 3.89×10^{11}	~ 4.97×10^{13}	~ 2.55×10^{16}	~ 3.26×10^{18}
$p = 2^{-10}$ (1 in 1024)	~ 1.24×10^{13}	~ 1.59×10^{15}	~ 8.15×10^{17}	~ 1.04×10^{20}
$p = 0.5$	~ 3.98×10^{14}	~ 3.60×10^{16}	~ 2.61×10^{19}	~ 2.36×10^{21}

Annex C

(informative)

Documentation summary

Table C.1 summarizes the documentation needed for compliance to this standard.

Table C.1—Documentation summary

Doc #	Documentation description	See
1	Provide documentation to the end-user about the cryptographic unit.	4.1
2	Describe how the plaintext record formatter generates plaintext records from host records.	4.3
3	Describe how the plaintext record de-formatter generates host records from plaintext records.	4.4
4	Disclose whether the cryptographic unit validates the MAC before returning any plaintext	4.6.2
5	Define the special signal *FAIL* and describe how the host and/or controller receive such a signal. The special signal *FAIL* should identify the host records that failed the MAC validation.	4.6.2
6	If the cryptographic unit is capable of returning plaintext before validating the MAC, then define the special signal *PASS*, describe how the host and/or controller receive such a signal, and define limits for the number of host records and bytes of plaintext that the cryptographic may return before checking the MAC.	4.6.2
7	If a cryptographic unit supports ordering verification, then specify the methods for enabling or disabling this functionality, and specify how the cryptographic unit notifies the host and/or controller of inconsistent IV or AAD ordering, and how to recover, if possible.	4.6.3
8	Describe all cryptographic parameters used by the cryptographic unit.	4.7
9	Specify the parameter limits for the cryptographic unit, if different from those in Table 2	5.1
10	If the cryptographic unit supports CBC-HMAC, then describe the format of the AAD and the method used to determine where the AAD ends and the CBC-IV starts	5.4
11	If the cryptographic unit supports XTS-HMAC, then describe the format of the AAD and the method used to determine where the AAD ends and the Tweak starts.	5.5
12	Describe the RBG employed by the cryptographic unit, including algorithms and descriptions of sources of randomness.	6.1
13	If the cryptographic unit supports cryptographic key entry or export, then specify the supported cryptographic key entry and export methods.	6.2
14	If the cryptographic unit supports key wrapping, then describe all key wrapping routines that the cryptographic unit supports.	6.4
15	Describe the format of the IV and the cryptographic unit's mechanism for generating each IV.	6.5.3.3
16	The cryptographic unit may clear its encryption session state based on a command received from the host. Describe such a command, if supported.	6.5.3.4
17	When creating unique IVs within a self-contained group, describe how the system prevents reuse of the same IV between any two cryptographic units within the self-contained group and how the cryptographic units are uniquely identified.	6.6

Annex D

(informative)

Test vectors

D.1 General

A cryptographic unit should test its cryptographic functions using test vectors included within this standard and/or within the reference documents. Table D.1 shows the recommended test vectors for each chosen cryptographic mode:

Table D.1—Recommended test vectors

Algorithm	Recommended test vectors
CCM-128-AES-256 (see 5.2)	See D.2
GCM-128-AES-256 (see 5.3)	See D.3
CBC-AES-256-HMAC-SHA-1 (see 5.4)	See D.4
CBC-AES-256-HMAC-SHA-256 (see 5.4)	See D.4
CBC-AES-256-HMAC-SHA-512 (see 5.4)	See D.4
XTS-AES-256-HMAC-SHA-512 (see 5.5)	See D.5

For all of the test vectors, the following abbreviations apply:

AAD	additionally authenticated data
CIV	CBC-IV for CBC-HMAC modes (see 5.4)
CTX	Ciphertext record
DUS	Data unit sequence number
HMK	160, 256, or 512-bit HMAC key
IV	Initialization vector
KEY	256-bit AES key
KEY1	(XTS only) first 256 bits of the cipher key
KEY2	(XTS only) second 256 bits of the cipher key
N/A	not applicable
NON	Nonce IV for CBC-HMAC modes (see 5.4)
PTX	Plaintext record
RPT	Repeat the previous AAD a given number of times
TAG	MAC (message authentication code)

For readability, the examples explicitly parse the XTS-AES-256 key into Key1 and Key2. HMK is the XTS-AES-256 HMAC key.

All numbers within the XTS test vectors are in little-endian bit order (same as IEEE Std 1619). The base for these numbers is hexadecimal.

All numbers within all other test vectors are in big-endian bit order, in which the most significant byte is on the left. The base for these numbers is hexadecimal, except for the “RPT” field, which is in decimal. Within a particular test vector, if multiple lines start with the same prefix, these lines are concatenated.

For all test vectors, each pair of hexadecimal digits is grouped into a byte such that the left digit is the most significant and the right digit is the least significant.

For the CBC-HMAC test vectors (see D.4), use only the leftmost bits of HMK, according to the key size required for the algorithm. For example, CBC-AES-256-HMAC-SHA-1 uses the first 160-bits of HMK, and CBC-AES-256-HMAC-SHA-256 uses the first 256-bits of HMK.

D.2 CCM-128-AES-256 test vectors

D.2.1 CCM-128-AES-256 test vector 1

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
IV 000000000000000000000000
PTX 00000000000000000000000000000000
CTX c1944044c8e7aa95d2de9513c7f3dd8c
TAG 4b0a3e5e51f151eb0ffae7c43d010fdb
```

D.2.2 CCM-128-AES-256 test vector 2

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
IV 000000000000000000000000
AAD 00000000000000000000000000000000
TAG 904704e89fb216443cb9d584911fc3c2
```

D.2.3 CCM-128-AES-256 test vector 3

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
IV 000000000000000000000000
AAD 00000000000000000000000000000000
PTX 00000000000000000000000000000000
CTX c1944044c8e7aa95d2de9513c7f3dd8c
TAG 87314e9c1fa01abe6a6415943dc38521
```

D.2.4 CCM-128-AES-256 test vector 4

```
KEY fb7615b23d80891dd470980bc79584c8b2fb64ce60978f4d17fce45a49e830b7
IV dbd1a3636024b7b402da7d6f
PTX a845348ec8c5b5f126f50e76fefd1b1e
CTX cc881261c6a7fa72b96a1739176b277f
TAG 3472e1145f2c0cbe146349062cf0e423
```

D.2.5 CCM-128-AES-256 test vector 5

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
IV 101112131415161718191a1b
AAD 000102030405060708090a0b0c0d0e0f10111213
PTX 202122232425262728292a2b2c2d2e2f3031323334353637
CTX 04f883aeb3bd0730eaf50bb6de4fa2212034e4e41b0e75e5
TAG 9bba3f3a107f3239bd63902923f80371
```

D.2.6 CCM-128-AES-256 test vector 6

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
IV 101112131415161718191a1b
AAD 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
AAD 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
AAD 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
AAD 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
AAD 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
AAD a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
AAD c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
AAD e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
RPT 0256
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
CTX 04f883aeb3bd0730eaf50bb6de4fa2212034e4e41b0e75e577f6bf2422c0f6d2
TAG 3376d2cf256ef613c56454cbb5265834
```

D.2.7 CCM-128-AES-256 test vector 7

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
IV 101112131415161718191a1b
AAD 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 24d8a38e939d2710cad52b96fe6f82010014c4c43b2e55c557d69f0402e0d6f2
CTX 06c53d6cbd3f1c3c6de5dcdcad9fb74f25741dea741149fe4278a0cc24741e86
CTX 58cc0523b8d7838c60fb1de4b7c3941f5b26dea9322aa29656ec37ac18a9b108
CTX a6f38b7917f5a9c398838b22afbd17252e96694a9e6237964a0eae21c0a6e152
CTX 15a0e82022926be97268249599e456e05029c3ebc07d78fc5b4a0862e04e68c2
CTX 9514c7bdafc4b52e04833bf30622e4eb42504a44a9dcbc774752de7bb82891ad
CTX 1eba9dc3281422a8aba8654268d3d9c81705f4c5a531ef856df5609a159af738
CTX eb753423ed2001b8f20c23725f2bef18c409f7e52132341f27cb8f0e79894dd9
TAG ebb1fa9d28ccfe21bdfea7e6d91e0bab
```

D.2.8 CCM-128-AES-256 test vector 8

```
KEY fb7615b23d80891dd470980bc79584c8b2fb64ce6097878d17fce45a49e830b7
IV dbd1a3636024b7b402da7d6f
AAD 36
PTX a9
CTX 9d
TAG 3261b1cf931431e99a32806738ecbd2a
```

D.2.9 CCM-128-AES-256 test vector 9

```
KEY f8d476cfd646ea6c2384cb1c27d6195dfef1a9f37b9c8d21a79c21f8cb90d289
IV dbd1a3636024b7b402da7d6f
AAD 7bd859a247961a21823b380e9fe8b65082ba61d3
PTX 90ae61cf7baebd4cade494c54a29ae70269aec71
CTX 6c05313e45dc8ec10bea6c670bd94f31569386a6
TAG 8f3829e8e76ee23c04f566189e63c686
```

D.3 GCM-128-AES-256 test vectors

D.3.1 GCM-128-AES-256 test vector 1

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
```

```
IV 000000000000000000000000
PTX 00000000000000000000000000000000
CTX cea7403d4d606b6e074ec5d3baf39d18
TAG d0d1c8a799996bf0265b98b5d48ab919
```

D.3.2 GCM-128-AES-256 test vector 2

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
IV 000000000000000000000000
AAD 00000000000000000000000000000000
TAG 2d45552d8575922b3ca3cc538442fa26
```

D.3.3 GCM-128-AES-256 test vector 3

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
IV 000000000000000000000000
AAD 00000000000000000000000000000000
PTX 00000000000000000000000000000000
CTX cea7403d4d606b6e074ec5d3baf39d18
TAG ae9b1771dba9cf62b39be017940330b4
```

D.3.4 GCM-128-AES-256 test vector 4

```
KEY fb7615b23d80891dd470980bc79584c8b2fb64ce60978f4d17fce45a49e830b7
IV dbd1a3636024b7b402da7d6f
PTX a845348ec8c5b5f126f50e76fefd1b1e
CTX 5df5d1fabcbbdd051538252444178704
TAG 4c43cce5a574d8a88b43d4353bd60f9f
```

D.3.5 GCM-128-AES-256 test vector 5

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
IV 101112131415161718191a1b
AAD 000102030405060708090a0b0c0d0e0f10111213
PTX 202122232425262728292a2b2c2d2e2f3031323334353637
CTX 591b1ff272b43204868ffc7bc7d521993526b6fa32247c3c
TAG 7de12a5670e570d8cae624a16df09c08
```

D.3.6 GCM-128-AES-256 test vector 6

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
IV 101112131415161718191a1b
AAD 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
AAD 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
AAD 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
AAD 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
AAD 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
AAD a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
AAD c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
AAD e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
RPT 0256
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
CTX 591b1ff272b43204868ffc7bc7d521993526b6fa32247c3c4057f3eae7548cef
TAG a1de5536e97edddccd26eeb1b5ff7b32
```

D.3.7 GCM-128-AES-256 test vector 7

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
IV 101112131415161718191a1b
AAD 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX 793b3fd252941224a6afdc5be7f501b9150696da12045c1c6077d3cac774accf
CTX c3d530d848d665d81a49cbb500b88bbb624ae61d1667229c302dc6ff0bb4d70b
CTX dbbc8566d6f5b158da99a2ff2e01dda629b89c34ad1e5feba70e7aae4328289c
CTX 3629b0588350581ca8b97ccf1258fa3bbe2c5026047ba72648969cff8ba10ae3
CTX 0e05935df0c693741892b76faf67133abd2cf2031121bd8bb38127a4d2eedeea
CTX 13276494f402cd7c107fb3ec3b24784834338e55436287092ac4a26f5ea7ea4a
CTX d68d73151639b05b24e68b9816d1398376d8e4138594758db9ad3b409259b26d
CTX cfc06e722be987b3767f70a7b856b774b1ba2685b368091429fccb8dcdde09e4
TAG 87ec837abf532855b2cea169d6943fcd
```

D.3.8 GCM-128-AES-256 test vector 8

```
KEY fb7615b23d80891dd470980bc79584c8b2fb64ce6097878d17fce45a49e830b7
IV dbd1a3636024b7b402da7d6f
AAD 36
PTX a9
CTX 0a
TAG be987d009a4b349aa80cb9c4ebc1e9f4
```

D.3.9 GCM-128-AES-256 test vector 9

```
KEY f8d476cfd646ea6c2384cb1c27d6195dfef1a9f37b9c8d21a79c21f8cb90d289
IV dbd1a3636024b7b402da7d6f
AAD 7bd859a247961a21823b380e9fe8b65082ba61d3
PTX 90ae61cf7baebd4cade494c54a29ae70269aec71
CTX ce2027b47a843252013465834d75fd0f0729752e
TAG acd8833837ab0ede84f4748da8899c15
```

D.3.10 GCM-128-AES-256 test vector 10

```
KEY dbbc8566d6f5b158da99a2ff2e01dda629b89c34ad1e5feba70e7aae4328289c
IV cfc06e722be987b3767f70a7b856b774
PTX ce2027b47a843252013465834d75fd0f
CTX dc03e524830d30f88e197f3acace66ef
TAG 9984eff6905755d1836f2db04089634c
```

D.3.11 GCM-128-AES-256 test vector 11

```
KEY 0e05935df0c693741892b76faf67133abd2cf2031121bd8bb38127a4d2eedeea
IV 74b1ba2685b368091429fccb8dcdde09e4
AAD 7bd859a247961a21823b380e9fe8b65082ba61d3
PTX 90ae61cf7baebd4cade494c54a29ae70269aec71
CTX 6be65e56066c4056738c03fe2320974ba3f65e09
```

```
TAG 6108dc417bf32f7fb7554ae52f088f87
```

D.3.12 GCM-128-AES-256 test vector 12

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
IV 02cbbc7a03eb4de39d80d1ebc988bfdf
AAD 688e1aa984de926dc7b4c47f44
PTX a2aab3ad8b17acdda288426cd7c429b7ca86b7aca05809c70ce82db25711cb53
PTX 02eb2743b036f3d750d6cf0dc0acb92950d546db308f93b4ff244afa9dc72bcd
PTX 758d2c
CTX ee62552aebc0c3c7daae12bb6c32ca5a005f4a1aaab004ed0f0b30abbf15acf4
CTX c50c59662d4b4468419544e7f981973563ce556ae50859ee09b14d31a053986f
CTX 9ac89b
TAG 9cd0db936e26d44be974ba868285a2e1
```

D.4 CBC-AES-256-HMAC-SHA test vectors (including HMAC-SHA-1, HMAC-SHA-256, and HMAC-SHA-512)

D.4.1 CBC-AES-256-HMAC-SHA test vector 1

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
NON N/A
CIV 00000000000000000000000000000000
PTX 00000000000000000000000000000000
CTX dc95c078a2408989ad48a21492842087
HMAC-SHA-1
TAG 59bb230e817ad3f377d623d2ca97eeffd0fd467c
HMAC-SHA-256
TAG 2cf16e982f18a9009687c8a8bf26cfd31e66bdda7277008d9564dd4779511855
HMAC-SHA-512
TAG bf8b5d45be53465f09ed9a4f53c565f067b3138318195425dfc466856973170d
TAG f8414dceb7d1c8888a622de9ea480840193f8ebd94c34a26bb692a31568e3949
```

D.4.2 CBC-AES-256-HMAC-SHA test vector 2

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
AAD 00000000000000000000000000000000
NON N/A
CIV 00000000000000000000000000000000
HMAC-SHA-1
TAG 66040990c7992a2a00d037d0b8631c0db1785897
HMAC-SHA-256
TAG 853c7403937d8b6239569b184eb7993fc5f751aefcea28f2c863858e2d29c50b
HMAC-SHA-512
TAG 65e879d47df1def0af378d32e9f4fe3a824fb51e2143c03322def229361af3b1
TAG 7a724a3d653d05cb9f41f4b90d09e8e2886a78da48537d1cfa62977a82e7374e
```

D.4.3 CBC-AES-256-HMAC-SHA test vector 3

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
AAD 00000000000000000000000000000000
NON N/A
CIV 00000000000000000000000000000000
PTX 00000000000000000000000000000000
CTX dc95c078a2408989ad48a21492842087
HMAC-SHA-1
TAG d5adb529213cd69a9a3d69cf2d10b0b469d936fe
HMAC-SHA-256
TAG 16a65111bd8e5a0af5f001f7d9200d44252bcfe5dc34da42315b99213b9cbb4b
HMAC-SHA-512
TAG 6d63ccdc62d5d376cc86eb6a144568d04b8cdf28955509df10a4bbe8c734d5af
TAG 37e8e524d30fed83d324b8dedb06d86636baa67f85caac73cc993f00ecb92dec
```

D.4.4 CBC-AES-256-HMAC-SHA test vector 4

```
KEY fb7615b23d80891dd470980bc79584c8b2fb64ce60978f4d17fce45a49e830b7
HMK 1b07a0e93c1f4c3aadff671dd2611ac2fe22d34c6b6d8630c30dd44f41d49fe5
HMK ad0a3dbdd0f13ca27e6523c5e4e2ab12884741a1af9b95f3cf6c0aec3b68ba40
NON N/A
CIV dbd1a3636024b7b402da7d6fe3fb056e
PTX a845348ec8c5b5f126f50e76fefd1b1e
CTX fd057a7f6d17bd747aced7b6fc948567
HMAC-SHA-1
TAG 3bd64954b1b5b0a98ac3a6f95d2e5fe65b5377c0
HMAC-SHA-256
TAG 3e5530fb364c80696b1b2f69e8d0de064a3e07ad1a0b795f00fcdec1649cabcb
HMAC-SHA-512
TAG 444aec157e48e683626bf14d26c9bfd9515d5def34582c034f0c3311dd7d9753
TAG 591f3effe264b8cdfaf755177b8a020a47edb7331fef628523d708aefe09b0da
```

D.4.5 CBC-AES-256-HMAC-SHA test vector 5

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
HMK 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
HMK 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
AAD 000102030405060708090a0b0c0d0e0f10111213
NON N/A
CIV 101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f
CTX 7b626546c8d79cdeb66edef23b9b7d72
HMAC-SHA-1
TAG c7932ddb8fc2212b56b1207e81019b556f4bb7d9
HMAC-SHA-256
TAG 6b0fe0b40a41e32d2c61726a3d7834014a8ee07873ccfe0c23f3a9073b90b099
HMAC-SHA-512
TAG efac7480579348343d1e9af4fc6896968080439717c3b2c3e63013aa718261f0
TAG e3ee43c6fdb4372f020d64c9fee4bc7743cfd9262d3adf03aec4f8d99fd178e4
```

D.4.6 CBC-AES-256-HMAC-SHA test vector 6

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
HMK 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
HMK 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
AAD 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
AAD 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
AAD 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
AAD 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
AAD 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
AAD a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
AAD c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
AAD e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
NON N/A
CIV 101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
CTX 7b626546c8d79cdeb66edef23b9b7d723d5f9d5bc2a411f1eb448442250eeca2
HMAC-SHA-1
TAG 49dc3eacadcc028df2bf9a4598e3fec6624c8b38
HMAC-SHA-256
TAG c2ea45b50293d8f62d348ef23aec702268eb66bb3e2248eb9f71a5817709da2f
HMAC-SHA-512
TAG 8b2e672aacc78b6ff58c770fd0d6ed252201ebbae95dceec912c0cf3bf27171b
TAG 3627a6fefc3cc8b9f9e64b542b64c06ebb786f986cdc8296bac15111dbffa82f
```

D.4.7 CBC-AES-256-HMAC-SHA test vector 7

```
KEY 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
HMK 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
HMK 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
AAD 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
NON N/A
CIV 101112131415161718191a1b1c1d1e1f
PTX 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
PTX 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
PTX 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
PTX 606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f
PTX 808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
PTX a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
PTX c0c1c2c3c4c5c6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf
PTX e0e1e2e3e4e5e6e7e8e9eaebecedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
CTX c9cb7b3859e1a550bcbf11b624022c56c3ad1479e5ce7034d7a03c13d8fb9502
CTX 6f7254c50ce4ebd743486d00e09ddd8e873a7e98984ad43f57088c510e911700
CTX 6acfe2fef69b4010f0f05a93af7d3a93a02085780fd5acb3a4eb870933077752
CTX 2f2c18e310ac0c0c3766bea3e97f71996336e4831f3b411fb2700ddbab565673
CTX 315bf4ab73c7e11abac4d0cfc228f1ac60dd10f85f9c2ade46a9af5eacb6a24a
CTX 43839b942e71ca4ce2080a809a04a849105da07efbbb2f60b9c376e0354e2a27
CTX da1eaa5c7adea77890cc25b6bd48229e17ce518040ceb46a04fc7b62444e77b5
CTX aaf3dbf60a660a2b68ec640622716b07758d99a0f598a73ed8bdae74fa3aae2f
HMAC-SHA-1
TAG 2e08d65f81ff646ad05ab7aaf42903aa760e577a
HMAC-SHA-256
TAG ebfe6f31be473ab22b649602a77f7408508dfa50cad109cbc97f2fe5f8bb8583
HMAC-SHA-512
TAG 7b3262045211619428 44c0970391344cdac71ce0440325b02203b537dd930799
TAG 0e158541dfc52cfcf69d3e8085658de4c98bc030273bad369fdf28aaad40e63c
```

D.4.8 CBC-AES-256-HMAC-SHA test vector 8

```
KEY fb7615b23d80891dd470980bc79584c8b2fb64ce6097878d17fce45a49e830b7
HMK cc84a6cca8f97b8a5624071aec7d09e7cf5bdaff239d467270f9716ba234d109
HMK ac60cf491d5105fc60fc5804c6474bc35cf9ead9123da80f649ca15a98a243d6
AAD 7bd859a2
```

```
NON N/A
CIV dbd1a3636024b7b402da7d6f54a67dc8
PTX 90ae61cf7baebd4cade494c54a29ae70
CTX 6cd763ff6144ede649c486f9404a5307
HMAC-SHA-1
TAG efc87d364ccab9d4bdc241185f1d847e2e16d8c4
HMAC-SHA-256
TAG f130415f56372bcd17250339d82118ca347be4cfff9f69181757cf5e98b0a775
HMAC-SHA-512
TAG 1604c3afb72546c2f6a9135df46ae799fdae4d9f5a87fdffd552016c5e4ed98a
TAG 393b62822df55b076e3dc6f9668234919bbdcc99f2b40379754cc6ac30c97250
```

D.4.9 CBC-AES-256-HMAC-SHA test vector 9

```
KEY 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
NON 00000000000000000000000000000000
CIV dc95c078a2408989ad48a2149284208 7
PTX 00000000000000000000000000000000
CTX 08c374848c228233c2b34f332bd2e9d3
HMAC-SHA-1
TAG dedf216e04f467eaad1e5a72b6a7c962c8281f13
HMAC-SHA-256
TAG 1f4dd7b6d7436b5b7d325c0c2411ed4fc02c101949eb8269e8166e8c6325e858
HMAC-SHA-512
TAG d8677480b0466345b3c32baa2c2b502fb3bfba01e759c4d1da04ca7c20dd9e55
TAG 00b3675d0e78e080125b68fd0c584ff3144b1e155a1136785ad723f3c69e23b5
```

D.5 XTS-AES-256-HMAC-SHA-512 test vectors

D.5.1 XTS-AES-256-HMAC-SHA-512 test vector 1

```
Key1 2718281828459045235360287471352662497757247093699959574966967627
Key2 3141592653589793238462643383279502884197169399375105820974944592
HMK 0000000000000000000000000000000000000000000000000000000000000000
HMK 0000000000000000000000000000000000000000000000000000000000000000
DUS ff000000000000000000000000000000
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
TAG 1c7105d3c1e8e235ffb013d5e8023729a35cdeacc16af1d7f5f0fec6c036b167
TAG 871649687c5692aaa0ada9773671939bbce2a3d15dcae43671aa6ca5f3a96a6f
```

D.5.2 XTS-AES-256-HMAC-SHA-512 test vector 2

```
Key1 2718281828459045235360287471352662497757247093699959574966967627
Key2 3141592653589793238462643383279502884197169399375105820974944592
HMK 1b07a0e93c1f4c3aadff671dd2611ac2fe22d34c6b6d8630c30dd44f41d49fe5
HMK ad0a3dbdd0f13ca27e6523c5e4e2ab12884741a1af9b95f3cf6c0aec3b68ba40
DUS ffff0000000000000000000000000000
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
```

```
TAG ecabc09097f1401bc289548a9b932bf197a1a7002665f36529e5e137395facc9
TAG 7133399c65a05f15cc81abc8067155ccaabd6fa64f744cb1d987d29100c7f523
```

D.5.3 XTS-AES-256-HMAC-SHA-512 test vector 3

```
Key1 2718281828459045235360287471352662497757247093699959574966967627
Key2 3141592653589793238462643383279502884197169399375105820974944592
HMK 072126bc492870f666b25023a548a9154b64d06f890ba3542b5198466c60c53d
HMK b4763ddda4de7bbc469113a8cd9196e064ff86b04d1cbbfdfdc305998402756d
AAD 636975712065685420e776f7262206b706d756a20786f6674207265766f2073
DUS ffffff00000000000000000000000000
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
CTX 97b1fa30a12fc1f65ba22905031539971a10f2f36c321bb51331cdefb39e3964
CTX c7ef079994f5b69b2edd83a71ef549971ee93f44eac3938fcdd61d01fa71799d
CTX a3a8091c4c48aa9ed263ff0749df95d44fef6a0bb578ec69456aa5408ae32c7a
CTX f08ad7ba8921287e3bbee31b767be06a0e705c864a769137df28292283ea81a2
CTX 480241b44d9921cdbec1bc28dc1fda114bd8e5217ac9d8ebafa720e9da4f9ace
CTX 231cc949e5b96fe76ffc21063fddc83a6b8679c00d35e09576a875305bed5f36
CTX ed242c8900dd1fa965bc950dfce09b132263a1eef52dd6888c309f5a7d712826
TAG a9fe02bbb70c062c93d958bc32936609a25a1ffa2dcd9f33aee88be73d943d4f
TAG dcbd459c0ecb0111c9c74cfcf2d5104f5f8262ae52444d6e744d8046f73ec7f2
```

D.5.4 XTS-AES-256-HMAC-SHA-512 test vector 4

```
Key1 2718281828459045235360287471352662497757247093699959574966967627
Key2 3141592653589793238462643383279502884197169399375105820974944592
HMK e19c148c56a3aa6737471aaba4909f06a17705e98bb8ee347e253c26cbf00cc5
HMK 3147ec26beb88413da0268d39bb4a707678277a0c927c10f565496d0fe3349d5
AAD e3e220f1f7f8ef20f9e820ece520e1e9ed20e6ea20e0ea20ecf4faf220f4e2f9
AAD 20e7e1e5f8e420f0e7eee3e420f9f6f6e420ebea2e0d0a
DUS ffffffff000000000000000000000000
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
TAG a6babc0886b9f7c7f16844449dc6fa549d4909969dab34f85287cd5a76bc6c41
TAG d58f3436f0654cad9987e04b95d54900d2a3e09c5264041941b5b56ba26cd7c2
```

D.5.5 XTS-AES-256-HMAC-SHA-512 test vector 5

```
Key1 2718281828459045235360287471352662497757247093699959574966967627
Key2 3141592653589793238462643383279502884197169399375105820974944592
HMK 6273d67c8fd3f0b06d1801507f3e42c06dd6d2a831914ad9439790fa9525a349
HMK da015634fee3bc417e41c3012174806c3242ae474e66c9f81f597ebd4b7ca2ca
AAD 4f6e63652075706f6e20612074696d652c20746865726520776173206120 6e6f
AAD 6e636f6e666f726d696e672073706172726f772077686f20646563696465640
AAD 6e6f7420746f20666c7920736f75746820666f722074686520776 96e7465722e
AAD 20736f756e64732e2054686520636174206c6561726e6420617761792207468
AAD 65206d616e7572652c20666f756e6420746865206368697270696e6720737061
AAD 72726f7720616e642070726f6d70746c792061746520686962e0a0a54484520
AAD 4d4f52414c204f46205448452053544f52590a0a312e2045766572796f6e6520
AAD 77686f20736869747320f6e20796f75206973206e6f74206e65636573736172
AAD 696c7920796f757220656e656d792e0a322e2045766572796f6e652077686f20
AAD 6765747320796f75206f7574206f66207368697420697320 6e6f74206e656365
AAD 73736172696c7920796f757220667269656e642e0a332e20416e642c20696620
AAD 796f75277265207761726d20616e642068617070792069 6e20612070696c6520
AAD 6f6620736869742c206b65657020796f7572206d6f75746820736875742100a0a
DUS ffffffff0000000000000000000000
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 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
TAG 0664e417f6740411cc10c55d6a8c6f43b7ad21f95f0f6b4751b6049990d13136
TAG 8fef3f1b42e172fbbec6b8133fdcbb8dccf3fed9c345818dc0ae11ace07e0c43
```