



# PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40

## Committee Specification 01

16 September 2014

### Specification URIs

#### This version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.doc>  
(Authoritative)  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.html>  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.pdf>

#### Previous version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd02/pkcs11-hist-v2.40-csprd02.doc>  
(Authoritative)  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd02/pkcs11-hist-v2.40-csprd02.html>  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd02/pkcs11-hist-v2.40-csprd02.pdf>

#### Latest version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.doc> (Authoritative)  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.pdf>

#### Technical Committee:

OASIS PKCS 11 TC

#### Chairs:

Robert Griffin ([robert.griffin@rsa.com](mailto:robert.griffin@rsa.com)), EMC Corporation  
Valerie Fenwick ([valerie.fenwick@oracle.com](mailto:valerie.fenwick@oracle.com)), Oracle

#### Editors:

Susan Gleeson ([susan.gleeson@oracle.com](mailto:susan.gleeson@oracle.com)), Oracle  
Chris Zimman ([chris@wmpp.com](mailto:chris@wmpp.com)), Individual

#### Related work:

This specification is related to:

- *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*. Edited by Susan Gleeson and Chris Zimman. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40*. Edited by Susan Gleeson and Chris Zimman. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40*. Edited by John Leiseboer and Robert Griffin. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.

- *PKCS #11 Cryptographic Token Interface Profiles Version 2.40*. Edited by Tim Hudson. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html>.

**Abstract:**

This document defines mechanisms for PKCS #11 that are no longer in general use.

**Status:**

This document was last revised or approved by the OASIS PKCS 11 TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=pkcs11#technical](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=pkcs11#technical).

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC’s public comment list, after subscribing to it by following the instructions at the “[Send A Comment](#)” button on the TC’s web page at <https://www.oasis-open.org/committees/pkcs11/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<https://www.oasis-open.org/committees/pkcs11/ipr.php>).

**Citation format:**

When referencing this specification the following citation format should be used:

**[PKCS11-Hist-v2.40]**

*PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40*. Edited by Susan Gleeson and Chris Zimman. 16 September 2014. OASIS Committee Specification 01. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.html>. Latest version: <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>.

---

# Notices

Copyright © OASIS Open 2014. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark> for above guidance.

---

# Table of Contents

1	Introduction .....	8
1.1	Description of this Document .....	8
1.2	Terminology .....	8
1.3	Definitions .....	8
1.4	Normative References .....	9
1.5	Non-Normative References .....	9
2	Mechanisms .....	12
2.1	PKCS #11 Mechanisms .....	12
2.2	FORTEZZA timestamp .....	15
2.3	KEA .....	15
2.3.1	Definitions .....	15
2.3.2	KEA mechanism parameters .....	15
2.3.3	KEA public key objects .....	16
2.3.4	KEA private key objects .....	17
2.3.5	KEA key pair generation .....	17
2.3.6	KEA key derivation .....	18
2.4	RC2 .....	19
2.4.1	Definitions .....	19
2.4.2	RC2 secret key objects .....	19
2.4.3	RC2 mechanism parameters .....	20
2.4.4	RC2 key generation .....	21
2.4.5	RC2-ECB .....	21
2.4.6	RC2-CBC .....	22
2.4.7	RC2-CBC with PKCS padding .....	22
2.4.8	General-length RC2-MAC .....	23
2.4.9	RC2-MAC .....	23
2.5	RC4 .....	24
2.5.1	Definitions .....	24
2.5.2	RC4 secret key objects .....	24
2.5.3	RC4 key generation .....	24
2.5.4	RC4 mechanism .....	25
2.6	RC5 .....	25
2.6.1	Definitions .....	25
2.6.2	RC5 secret key objects .....	25
2.6.3	RC5 mechanism parameters .....	26
2.6.4	RC5 key generation .....	27
2.6.5	RC5-ECB .....	27
2.6.6	RC5-CBC .....	28
2.6.7	RC5-CBC with PKCS padding .....	28
2.6.8	General-length RC5-MAC .....	29
2.6.9	RC5-MAC .....	29
2.7	General block cipher .....	30
2.7.1	Definitions .....	30

2.7.2 DES secret key objects .....	31
2.7.3 CAST secret key objects .....	32
2.7.4 CAST3 secret key objects .....	32
2.7.5 CAST128 (CAST5) secret key objects .....	33
2.7.6 IDEA secret key objects .....	33
2.7.7 CDMF secret key objects .....	34
2.7.8 General block cipher mechanism parameters.....	34
2.7.9 General block cipher key generation.....	34
2.7.10 General block cipher ECB.....	35
2.7.11 General block cipher CBC.....	35
2.7.12 General block cipher CBC with PKCS padding.....	36
2.7.13 General-length general block cipher MAC .....	37
2.7.14 General block cipher MAC .....	37
2.8 SKIPJACK.....	38
2.8.1 Definitions.....	38
2.8.2 SKIPJACK secret key objects .....	38
2.8.3 SKIPJACK Mechanism parameters .....	39
2.8.4 SKIPJACK key generation .....	41
2.8.5 SKIPJACK-ECB64 .....	41
2.8.6 SKIPJACK-CBC64 .....	41
2.8.7 SKIPJACK-OFB64 .....	41
2.8.8 SKIPJACK-CFB64.....	42
2.8.9 SKIPJACK-CFB32.....	42
2.8.10 SKIPJACK-CFB16.....	42
2.8.11 SKIPJACK-CFB8.....	43
2.8.12 SKIPJACK-WRAP .....	43
2.8.13 SKIPJACK-PRIVATE-WRAP .....	43
2.8.14 SKIPJACK-RELAYX.....	43
2.9 BATON.....	43
2.9.1 Definitions.....	43
2.9.2 BATON secret key objects .....	44
2.9.3 BATON key generation .....	44
2.9.4 BATON-ECB128 .....	45
2.9.5 BATON-ECB96.....	45
2.9.6 BATON-CBC128 .....	45
2.9.7 BATON-COUNTER .....	46
2.9.8 BATON-SHUFFLE .....	46
2.9.9 BATON WRAP .....	46
2.10 JUNIPER.....	46
2.10.1 Definitions.....	46
2.10.2 JUNIPER secret key objects .....	47
2.10.3 JUNIPER key generation .....	47
2.10.4 JUNIPER-ECB128 .....	48
2.10.5 JUNIPER-CBC128 .....	48
2.10.6 JUNIPER-COUNTER .....	48

2.10.7 JUNIPER-SHUFFLE .....	48
2.10.8 JUNIPER WRAP .....	49
2.11 MD2 .....	49
2.11.1 Definitions .....	49
2.11.2 MD2 digest .....	49
2.11.3 General-length MD2-HMAC .....	49
2.11.4 MD2-HMAC .....	50
2.11.5 MD2 key derivation .....	50
2.12 MD5 .....	50
2.12.1 Definitions .....	50
2.12.2 MD5 Digest .....	51
2.12.3 General-length MD5-HMAC .....	51
2.12.4 MD5-HMAC .....	51
2.12.5 MD5 key derivation .....	51
2.13 FASTHASH .....	52
2.13.1 Definitions .....	52
2.13.2 FASTHASH digest .....	52
2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD) .....	52
2.14.1 Definitions .....	52
2.14.2 Password-based encryption/authentication mechanism parameters .....	53
2.14.3 MD2-PBE for DES-CBC .....	53
2.14.4 MD5-PBE for DES-CBC .....	53
2.14.5 MD5-PBE for CAST-CBC .....	54
2.14.6 MD5-PBE for CAST3-CBC .....	54
2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC) .....	54
2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC) .....	54
2.15 PKCS #12 password-based encryption/authentication mechanisms .....	55
2.15.1 Definitions .....	55
2.15.2 SHA-1-PBE for 128-bit RC4 .....	55
2.15.3 SHA-1_PBE for 40-bit RC4 .....	56
2.15.4 SHA-1_PBE for 128-bit RC2-CBC .....	56
2.15.5 SHA-1_PBE for 40-bit RC2-CBC .....	56
2.16 RIPE-MD .....	56
2.16.1 Definitions .....	56
2.16.2 RIPE-MD 128 Digest .....	57
2.16.3 General-length RIPE-MD 128-HMAC .....	57
2.16.4 RIPE-MD 128-HMAC .....	57
2.16.5 RIPE-MD 160 .....	57
2.16.6 General-length RIPE-MD 160-HMAC .....	58
2.16.7 RIPE-MD 160-HMAC .....	58
2.17 SET .....	58
2.17.1 Definitions .....	58
2.17.2 SET mechanism parameters .....	58
2.17.3 OAEP key wrapping for SET .....	59
2.18 LYNKS .....	59

2.18.1 Definitions .....	59
2.18.2 LYNKS key wrapping .....	59
3 PKCS #11 Implementation Conformance .....	60
Appendix A. Acknowledgments .....	61
Appendix B. Manifest constants .....	64
Appendix C. Revision History .....	67

---

# 1 Introduction

## 1.1 Description of this Document

This document defines historical PKCS#11 mechanisms, that is, mechanisms that were defined for earlier versions of PKCS #11 but are no longer in general use

All text is normative unless otherwise labeled.

## 1.2 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

## 1.3 Definitions

For the purposes of this standard, the following definitions apply. Please refer to [PKCS#11-Base] for further definitions

<b>BATON</b>	MISSI's BATON block cipher.
<b>CAST</b>	Entrust Technologies' proprietary symmetric block cipher
<b>CAST3</b>	Entrust Technologies' proprietary symmetric block cipher
<b>CAST5</b>	Another name for Entrust Technologies' symmetric block cipher CAST128. CAST128 is the preferred name.
<b>CAST128</b>	Entrust Technologies' symmetric block cipher.
<b>CDMF</b>	Commercial Data Masking Facility, a block encipherment method specified by International Business Machines Corporation and based on DES.
<b>CMS</b>	Cryptographic Message Syntax (see RFC 3369)
<b>DES</b>	Data Encryption Standard, as defined in FIPS PUB 46-3
<b>ECB</b>	Electronic Codebook mode, as defined in FIPS PUB 81.
<b>FASTHASH</b>	MISSI's FASTHASH message-digesting algorithm.
<b>IDEA</b>	Ascom Systec's symmetric block cipher.
<b>IV</b>	Initialization Vector.
<b>JUNIPER</b>	MISSI's JUNIPER block cipher.
<b>KEA</b>	MISSI's Key Exchange Algorithm.
<b>LYNKS</b>	A smart card manufactured by SPYRUS.
<b>MAC</b>	Message Authentication Code
<b>MD2</b>	RSA Security's MD2 message-digest algorithm, as defined in RFC 6149.
<b>MD5</b>	RSA Security's MD5 message-digest algorithm, as defined in RFC 1321.



38	<b>PRF</b>	Pseudo random function.
39	<b>RSA</b>	The RSA public-key cryptosystem.
40	<b>RC2</b>	RSA Security's RC2 symmetric block cipher.
41	<b>RC4</b>	RSA Security's proprietary RC4 symmetric stream cipher.
42	<b>RC5</b>	RSA Security's RC5 symmetric block cipher.
43	<b>SET</b>	<b>The Secure Electronic Transaction protocol.</b>
44	<b>SHA-1</b>	The (revised) Secure Hash Algorithm with a 160-bit message digest, as defined in FIPS PUB 180-2.
46	<b>SKIPJACK</b>	MISSI's SKIPJACK block cipher.

## 48 1.4 Normative References

49	<b>[PKCS #11-Base]</b>	<i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i> Edited by Susan Gleeson and Chris Zimman. 16 September 2014. OASIS Committee Specification 01. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/cs01/pkcs11-base-v2.40-cs01.html">http://docs.oasis-open.org/pkcs11/pkcs11- base/v2.40/cs01/pkcs11-base-v2.40-cs01.html</a> . Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html">http://docs.oasis- open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html</a> .
54	<b>[PKCS #11-Curr]</b>	<i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40.</i> Edited by Susan Gleeson and Chris Zimman. 16 September 2014. OASIS Committee Specification 01. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/cs01/pkcs11-curr-v2.40-cs01.html">http://docs.oasis-open.org/pkcs11/pkcs11- curr/v2.40/cs01/pkcs11-curr-v2.40-cs01.html</a> . Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html">http://docs.oasis- open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html</a> .
59	<b>[PKCS #11-Prof]</b>	<i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i> Edited by Tim Hudson. 16 September 2014. OASIS Committee Specification 01. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/cs01/pkcs11-profiles-v2.40-cs01.html">http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/cs01/pkcs11-profiles- v2.40-cs01.html</a> . Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11- profiles/v2.40/pkcs11-profiles-v2.40.html</a> .
64	<b>[RFC2119]</b>	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> .

## 67 1.5 Non-Normative References

68	<b>[ANSI C]</b>	ANSI/ISO. <i>American National Standard for Programming Languages – C</i> . 1990
69	<b>[ANSI X9.31]</b>	Accredited Standards Committee X9. <i>Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)</i> . 1998.
71	<b>[ANSI X9.42]</b>	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography</i> . 2003
74	<b>[ANSI X9.62]</b>	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)</i> . 1998
76	<b>[CC/PP]</b>	G. Klyne, F. Reynolds, C. , H. Ohto, J. Hjelm, M. H. Butler, L. Tran, Editors, W3C. <i>Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies</i> . 2004, URL: <a href="http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/">http://www.w3.org/TR/2004/REC-CCPP-struct- vocab-20040115/</a>
80	<b>[CDPD]</b>	Ameritech Mobile Communications et al. <i>Cellular Digital Packet Data System Specifications: Part 406: Airlink Security</i> . 1993
82	<b>[FIPS PUB 46-3]</b>	NIST. <i>FIPS 46-3: Data Encryption Standard (DES)</i> . October 26, 2999. URL: <a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>

- [FIPS PUB 81] NIST. *FIPS 81: DES Modes of Operation*. December 1980. URL: <http://csrc.nist.gov/publications/fips/index.html>
- [FIPS PUB 113] NIST. *FIPS 113: Computer Data Authentication*. May 30, 1985. URL: <http://csrc.nist.gov/publications/fips/index.html>
- [FIPS PUB 180-2] NIST. *FIPS 180-2: Secure Hash Standard*. August 1, 2002. URL: <http://csrc.nist.gov/publications/fips/index.html>
- [FORTEZZA CIPG] NSA, Workstation Security Products. *FORTEZZA Cryptologic Interface Programmers Guide, Revision 1.52*. November 1985
- [GCS-API] X/Open Company Ltd. *Generic Cryptographic Service API (GCS-API), Base – Draft 2*. February 14, 1995.
- [ISO/IEC 7816-1] ISO/IEC 7816-1:2011. *Identification Cards – Integrated circuit cards -- Part 1: Cards with contacts -- Physical Characteristics*. 2011 URL: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=54089](http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089).
- [ISO/IEC 7816-4] ISO/IEC 7816-4:2013. *Identification Cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*. 2013. URL: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=54550](http://www.iso.org/iso/catalogue_detail.htm?csnumber=54550).
- [ISO/IEC 8824-1] ISO/IEC 8824-1:2008. *Abstract Syntax Notation One (ASN.1): Specification of Base Notation*. 2002. URL: [http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=54012](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012)
- [ISO/IEC 8825-1] ISO/IEC 8825-1:2008. *Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*. 2008. URL: [http://www.iso.org/iso/home/store/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60)
- [ISO/IEC 9594-1] ISO/IEC 9594-1:2008. *Information Technology – Open System Interconnection – The Directory: Overview of Concepts, Models and Services*. 2008. URL: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=53364](http://www.iso.org/iso/catalogue_detail.htm?csnumber=53364)
- [ISO/IEC 9594-8] ISO/IEC 9594-8:2008. *Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks*. 2008 URL: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=53372](http://www.iso.org/iso/catalogue_detail.htm?csnumber=53372)
- [ISO/IEC 9796-2] ISO/IEC 9796-2:2010. *Information Technology – Security Techniques – Digital Signature Scheme Giving Message Recovery – Part 2: Integer factorization based mechanisms*. 2010. URL: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=54788](http://www.iso.org/iso/catalogue_detail.htm?csnumber=54788)
- [Java MIDP] Java Community Process. *Mobile Information Device Profile for Java 2 Micro Edition*. November 2002. URL: <http://jcp.org/jsr/detail/118.jsp>
- [MeT-PTD] MeT. *MeT PTD Definition – Personal Trusted Device Definition, Version 1.0*. February 2003. URL: <http://www.mobiletransaction.org>
- [PCMCIA] Personal Computer Memory Card International Association. *PC Card Standard, Release 2.1*. July 1993.
- [PKCS #1] RSA Laboratories. *RSA Cryptography Standard, v2.1*. June 14, 2002 URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>
- [PKCS #3] RSA Laboratories. *Diffie-Hellman Key-Agreement Standard, v1.4*. November 1993.
- [PKCS #5] RSA Laboratories. *Password-Based Encryption Standard, v2.0*. March 26, 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf>
- [PKCS #7] RSA Laboratories. *Cryptographic Message Syntax Standard, v1.6*. November 1997 URL : <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v16.pdf>

<b>[PKCS #8]</b>	RSA Laboratories. <i>Private-Key Information Syntax Standard</i> , v1.2. November 1993. URL : <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn</a>
<b>[PKCS #11-UG]</b>	<i>PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40</i> . Edited by John Leiseboer and Robert Griffin. 16 September 2014. OASIS Committee Note 01. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn01/pkcs11-ug-v2.40-cn01.html">http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn01/pkcs11-ug-v2.40-cn01.html</a> . Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html</a> .
<b>[PKCS #12]</b>	RSA Laboratories. <i>Personal Information Exchange Syntax Standard</i> , v1.0. June 1999. URL: <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf</a>
<b>[RFC 1321]</b>	R. Rivest. <i>RFC 1321: The MD5 Message-Digest Algorithm</i> . MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. URL: <a href="http://www.rfc-editor.org/rfc/rfc1321.txt">http://www.rfc-editor.org/rfc/rfc1321.txt</a>
<b>[RFC 3369]</b>	R. Houseley. <i>RFC 3369: Cryptographic Message Syntax (CMS)</i> . August 2002. URL: <a href="http://www.rfc-editor.org/rfc/rfc3369.txt">http://www.rfc-editor.org/rfc/rfc3369.txt</a>
<b>[RFC 6149]</b>	S. Turner and L. Chen. <i>RFC 6149: MD2 to Historic Status</i> . March, 2011. URL: <a href="http://www.rfc-editor.org/rfc/rfc6149.txt">http://www.rfc-editor.org/rfc/rfc6149.txt</a>
<b>[SEC-1]</b>	Standards for Efficient Cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography</i> . Version 1.0, September 20, 2000.
<b>[SEC-2]</b>	Standards for Efficient cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters</i> . Version 1.0, September 20, 2000.
<b>[TLS]</b>	IETF. <i>RFC 2246: The TLS Protocol Version 1.0</i> . January 1999. URL: <a href="http://ietf.org/rfc/rfc2256.txt">http://ietf.org/rfc/rfc2256.txt</a>
<b>[WIM]</b>	WAP. <i>Wireless Identity Module</i> . – WAP-260-WIP-20010712.a. July 2001. URL: <a href="http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf">http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf</a>
<b>[WPKI]</b>	WAP. <i>Wireless Application Protocol: Public Key Infrastructure Definition</i> . – WAP-217-WPKI-20010424-a. April 2001. URL: <a href="http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf">http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf</a>
<b>[WTLS]</b>	WAP. <i>Wireless Transport Layer Security Version</i> – WAP-261-WTLS-20010406-a. April 2001. URL: <a href="http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf">http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf</a>
<b>[X.500]</b>	ITU-T. <i>Information Technology – Open Systems Interconnection – The Directory: Overview of Concepts, Models and Services</i> . February 2001. (Identical to ISO/IEC 9594-1)
<b>[X.509]</b>	ITU-T. <i>Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks</i> . March 2000. (Identical to ISO/IEC 9594-8)
<b>[X.680]</b>	ITU-T. <i>Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation</i> . July 2002. (Identical to ISO/IEC 8824-1)
<b>[X.690]</b>	ITU-T. <i>Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)</i> . July 2002. (Identical to ISO/IEC 8825-1)

## 2 Mechanisms

### 2.1 PKCS #11 Mechanisms

A mechanism specifies precisely how a certain cryptographic process is to be performed. PKCS #11 implementations MAY use one or more mechanisms defined in this document.

The following table shows which Cryptoki mechanisms are supported by different cryptographic operations. For any particular token, of course, a particular operation MAY support only a subset of the mechanisms listed. There is also no guarantee that a token which supports one mechanism for some operation supports any other mechanism for any other operation (or even supports that same mechanism for any other operation). For example, even if a token is able to create RSA digital signatures with the **CKM\_RSA\_PKCS** mechanism, it may or may not be the case that the same token MAY also perform RSA encryption with **CKM\_RSA\_PKCS**.

Table 1, Mechanisms vs. Functions

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_FORTEZZA_TIMESTAMP		X <sup>2</sup>					
CKM_KEA_KEY_PAIR_GEN					X		
CKM_KEA_KEY_DERIVE							X
CKM_RC2_KEY_GEN					X		
CKM_RC2_ECB	X					X	
CKM_RC2_CBC	X					X	
CKM_RC2_CBC_PAD	X					X	
CKM_RC2_MAC_GENERAL		X					
CKM_RC2_MAC		X					
CKM_RC4_KEY_GEN					X		
CKM_RC4	X						
CKM_RC5_KEY_GEN					X		
CKM_RC5_ECB	X					X	
CKM_RC5_CBC	X					X	
CKM_RC5_CBC_PAD	X					X	
CKM_RC5_MAC_GENERAL		X					
CKM_RC5_MAC		X					
CKM_DES_KEY_GEN					X		
CKM_DES_ECB	X					X	
CKM_DES_CBC	X					X	
CKM_DES_CBC_PAD	X					X	
CKM_DES_MAC_GENERAL		X					
CKM_DES_MAC		X					
CKM_CAST_KEY_GEN					X		
CKM_CAST_ECB	X					X	
CKM_CAST_CBC	X					X	
CKM_CAST_CBC_PAD	X					X	

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_CAST_MAC_GENERAL		X					
CKM_CAST_MAC		X					
CKM_CAST3_KEY_GEN					X		
CKM_CAST3_ECB	X					X	
CKM_CAST3_CBC	X					X	
CKM_CAST3_CBC_PAD	X					X	
CKM_CAST3_MAC_GENERAL		X					
CKM_CAST3_MAC		X					
CKM_CAST128_KEY_GEN (CKM_CAST5_KEY_GEN)					X		
CKM_CAST128_ECB (CKM_CAST5_ECB)	X					X	
CKM_CAST128_CBC (CKM_CAST5_CBC)	X					X	
CKM_CAST128_CBC_PAD (CKM_CAST5_CBC_PAD)	X					X	
CKM_CAST128_MAC_GENERAL (CKM_CAST5_MAC_GENERAL)		X					
CKM_CAST128_MAC (CKM_CAST5_MAC)		X					
CKM_IDEA_KEY_GEN					X		
CKM_IDEA_ECB	X					X	
CKM_IDEA_CBC	X					X	
CKM_IDEA_CBC_PAD	X					X	
CKM_IDEA_MAC_GENERAL		X					
CKM_IDEA_MAC		X					
CKM_CDMF_KEY_GEN					X		
CKM_CDMF_ECB	X					X	
CKM_CDMF_CBC	X					X	
CKM_CDMF_CBC_PAD	X					X	
CKM_CDMF_MAC_GENERAL		X					
CKM_CDMF_MAC		X					
CKM_SKIPJACK_KEY_GEN					X		
CKM_SKIPJACK_ECB64	X						
CKM_SKIPJACK_CBC64	X						
CKM_SKIPJACK_OF64	X						
CKM_SKIPJACK_CFB64	X						
CKM_SKIPJACK_CFB32	X						
CKM_SKIPJACK_CFB16	X						
CKM_SKIPJACK_CFB8	X						
CKM_SKIPJACK_WRAP						X	
CKM_SKIPJACK_PRIVATE_WRAP						X	
CKM_SKIPJACK_RELAYX						X <sup>3</sup>	
CKM_BATON_KEY_GEN					X		
CKM_BATON_ECB128	X						
CKM_BATON_ECB96	X						

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_BATON_CBC128	X						
CKM_BATON_COUNTER	X						
CKM_BATON_SHUFFLE	X						
CKM_BATON_WRAP						X	
CKM_JUNIPER_KEY_GEN					X		
CKM_JUNIPER_ECB128	X						
CKM_JUNIPER_CBC128	X						
CKM_JUNIPER_COUNTER	X						
CKM_JUNIPER_SHUFFLE	X						
CKM_JUNIPER_WRAP						X	
CKM_MD2				X			
CKM_MD2_HMAC_GENERAL		X					
CKM_MD2_HMAC		X					
CKM_MD2_KEY_DERIVATION							X
CKM_MD5				X			
CKM_MD5_HMAC_GENERAL		X					
CKM_MD5_HMAC		X					
CKM_MD5_KEY_DERIVATION							X
CKM_RIPEMD128				X			
CKM_RIPEMD128_HMAC_GENERAL		X					
CKM_RIPEMD128_HMAC		X					
CKM_RIPEMD160				X			
CKM_RIPEMD160_HMAC_GENERAL		X					
CKM_RIPEMD160_HMAC		X					
CKM_FASTHASH				X			
CKM_PBE_MD2_DES_CBC					X		
CKM_PBE_MD5_DES_CBC					X		
CKM_PBE_MD5_CAST_CBC					X		
CKM_PBE_MD5_CAST3_CBC					X		
CKM_PBE_MD5_CAST128_CBC (CKM_PBE_MD5_CAST5_CBC)					X		
CKM_PBE_SHA1_CAST128_CBC (CKM_PBE_SHA1_CAST5_CBC)					X		
CKM_PBE_SHA1_RC4_128					X		
CKM_PBE_SHA1_RC4_40					X		
CKM_PBE_SHA1_RC2_128_CBC					X		
CKM_PBE_SHA1_RC2_40_CBC					X		
CKM_PBA_SHA1_WITH_SHA1_HMAC					X		
CKM_KEY_WRAP_SET_OAEP						X	
CKM_KEY_WRAP_LYNKS						X	

<sup>1</sup> SR = SignRecover, VR = VerifyRecover.

<sup>2</sup> Single-part operations only.

<sup>3</sup> Mechanism MUST only be used for wrapping, not unwrapping.

The remainder of this section presents in detail the mechanisms supported by Cryptoki and the parameters which are supplied to them.

In general, if a mechanism makes no mention of the *ulMinKeyLen* and *ulMaxKeyLen* fields of the CK\_MECHANISM\_INFO structure, then those fields have no meaning for that particular mechanism.

## 2.2 FORTEZZA timestamp

The FORTEZZA timestamp mechanism, denoted **CKM\_FORTEZZA\_TIMESTAMP**, is a mechanism for single-part signatures and verification. The signatures it produces and verifies are DSA digital signatures over the provided hash value and the current time.

**It has no parameters.**

Constraints on key types and the length of data are summarized in the following table. The input and output data MAY begin at the same location in memory.

Table 2, FORTEZZA Timestamp: Key and Data Length

Function	Key type	Input Length	Output Length
C_Sign <sup>1</sup>	DSA private key	20	40
C_Verify <sup>1</sup>	DSA public key	20,40 <sup>2</sup>	N/A

<sup>1</sup> Single-part operations only

<sup>2</sup> Data length, signature length

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure specify the supported range of DSA prime sizes, in bits.

## 2.3 KEA

### 2.3.1 Definitions

This section defines the key type “CKK\_KEA” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

CKM\_KEA\_KEY\_PAIR\_GEN

CKM\_KEA\_KEY\_DERIVE

### 2.3.2 KEA mechanism parameters

#### 2.3.2.1 CK\_KEA\_DERIVE\_PARAMS; CK\_KEA\_DERIVE\_PARAMS\_PTR

**CK\_KEA\_DERIVE\_PARAMS** is a structure that provides the parameters to the **CKM\_KEA\_DERIVE** mechanism. It is defined as follows:

```
typedef struct CK_KEA_DERIVE_PARAMS {
    CK_BBOOL isSender;
    CK_ULONG ulRandomLen;
    CK_BYTE_PTR pRandomA;
    CK_BYTE_PTR pRandomB;
    CK_ULONG ulPublicDataLen;
    CK_BYTE_PTR pPublicData;
} CK_KEA_DERIVE_PARAMS;
```

The fields of the structure have the following meanings:



239                    *isSender*      Option for generating the key (called a TEK). The value  
240    is CK\_TRUE if the sender (originator) generates the  
241    TEK, CK\_FALSE if the recipient is regenerating the TEK

242                    *ulRandomLen*      the size of random Ra and Rb in bytes

243                    *pRandomA*      pointer to Ra data

244                    *pRandomB*      pointer to Rb data

245                    *ulPublicDataLen*      other party's KEA public key size

246                    *pPublicData*      pointer to other party's KEA public key value

247    **CK\_KEA\_DERIVE\_PARAMS\_PTR** is a pointer to a **CK\_KEA\_DERIVE\_PARAMS**.

### 248    2.3.3 KEA public key objects

249    KEA public key objects (object class **CKO\_PUBLIC\_KEY**, key type **CKK\_KEA**) hold KEA public keys.  
250    The following table defines the KEA public key object attributes, in addition to the common attributes  
251    defined for this object class:

252    Table 3, KEA Public Key Object Attributes

Attribute	Data type	Meaning
CKA_PRIME <sup>1,3</sup>	Big integer	Prime $p$ (512 to 1024 bits, in steps of 64 bits)
CKA_SUBPRIME <sup>1,3</sup>	Big integer	Subprime $q$ (160 bits)
CKA_BASE <sup>1,3</sup>	Big integer	Base $g$ (512 to 1024 bits, in steps of 64 bits)
CKA_VALUE <sup>1,4</sup>	Big integer	Public value $y$

253    <sup>1</sup> Refer to [PKCS #11-Base] table 15 for footnotes

254    The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the "KEA domain  
255    parameters".

256    The following is a sample template for creating a KEA public key object:

```

257    CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
258    CK_KEY_TYPE keyType = CKK_KEA;
259    CK_UTF8CHAR label[] = "A KEA public key object";
260    CK_BYTE prime[] = {...};
261    CK_BYTE subprime[] = {...};
262    CK_BYTE base[] = {...};
263    CK_BYTE value[] = {...};
264    CK_ATTRIBUTE template[] = {
265       {CKA_CLASS, &class, sizeof(class)},
266       {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
267       {CKA_TOKEN, &true, sizeof(true)},
268       {CKA_LABEL, label, sizeof(label)-1},
269       {CKA_PRIME, prime, sizeof(prime)},
270       {CKA_SUBPRIME, subprime, sizeof(subprime)},
271       {CKA_BASE, base, sizeof(base)},
272       {CKA_VALUE, value, sizeof(value)}
273    };

```

274



### 2.3.4 KEA private key objects

KEA private key objects (object class **CKO\_PRIVATE\_KEY**, key type **CKK\_KEA**) hold KEA private keys. The following table defines the KEA private key object attributes, in addition to the common attributes defined for this object class:

Table 4, KEA Private Key Object Attributes

Attribute	Data type	Meaning
CKA_PRIME <sup>1,4,6</sup>	Big integer	Prime $p$ (512 to 1024 bits, in steps of 64 bits)
CKA_SUBPRIME <sup>1,4,6</sup>	Big integer	Subprime $q$ (160 bits)
CKA_BASE <sup>1,4,6</sup>	Big integer	Base $g$ (512 to 1024 bits, in steps of 64 bits)
CKA_VALUE <sup>1,4,6,7</sup>	Big integer	Private value $x$

Refer to [PKCS #11-Base] table 15 for footnotes

The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the “KEA domain parameters”.

Note that when generating a KEA private key, the KEA parameters are *not* specified in the key’s template. This is because KEA private keys are only generated as part of a KEA key *pair*, and the KEA parameters for the pair are specified in the template for the KEA public key.

The following is a sample template for creating a KEA private key object:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_KEA;
CK_UTF8CHAR label[] = "A KEA private key object";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label) - 1},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_ID, id, sizeof(id)},
    {CKA_SENSITIVE, &true, sizeof(true)},
    {CKA_DERIVE, &true, sizeof(true)},
    {CKA_PRIME, prime, sizeof(prime)},
    {CKA_SUBPRIME, subprime, sizeof(subprime)},
    {CKA_BASE, base, sizeof(base)},
    {CKA_VALUE, value, sizeof(value)}
};
```

### 2.3.5 KEA key pair generation

The KEA key pair generation mechanism, denoted **CKM\_KEA\_KEY\_PAIR\_GEN**, generates key pairs for the Key Exchange Algorithm, as defined by NIST’s “SKIPJACK and KEA Algorithm Specification Version 2.0”, 29 May 1998.

It does not have a parameter.

The mechanism generates KEA public/private key pairs with a particular prime, subprime and base, as specified in the **CKA\_PRIME**, **CKA\_SUBPRIME**, and **CKA\_BASE** attributes of the template for the public

key. Note that this version of Cryptoki does not include a mechanism for generating these KEA domain parameters.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE** and **CKA\_VALUE** attributes to the new public key and the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_SUBPRIME**, **CKA\_BASE**, and **CKA\_VALUE** attributes to the new private key. Other attributes supported by the KEA public and private key types (specifically, the flags indicating which functions the keys support) MAY also be specified in the templates for the keys, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of KEA prime sizes, in bits.

### 2.3.6 KEA key derivation

The KEA key derivation mechanism, denoted **CKM\_DEA\_DERIVE**, is a mechanism for key derivation based on KEA, the Key Exchange Algorithm, as defined by NIST's "SKIPJACK and KEA Algorithm Specification Version 2.0", 29 May 1998.

It has a parameter, a **CK\_KEA\_DERIVE\_PARAMS** structure.

This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE** attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key type must be specified in the template.

As defined in the Specification, KEA MAY be used in two different operational modes: full mode and e-mail mode. Full mode is a two-phase key derivation sequence that requires real-time parameter exchange between two parties. E-mail mode is a one-phase key derivation sequence that does not require real-time parameter exchange. By convention, e-mail mode is designated by use of a fixed value of one (1) for the KEA parameter  $R_b$  (*pRandomB*).

The operation of this mechanism depends on two of the values in the supplied **CK\_KEA\_DERIVE\_PARAMS** structure, as detailed in the table below. Note that in all cases, the data buffers pointed to by the parameter structure fields *pRandomA* and *pRandomB* must be allocated by the caller prior to invoking **C\_DeriveKey**. Also, the values pointed to by *pRandomA* and *pRandomB* are represented as Cryptoki "Big integer" data (i.e., a sequence of bytes, most significant byte first).

Table 5, KEA Parameter Values and Operations

Value of boolean <i>isSender</i>	Value of big integer <i>pRandomB</i>	Token Action (after checking parameter and template values)
CK_TRUE	0	Compute KEA $R_a$ value, store it in <i>pRandomA</i> , return CKR_OK. No derived key object is created.
CK_TRUE	1	Compute KEA $R_a$ value, store it in <i>pRandomA</i> , derive key value using e-mail mode, create key object, return CKR_OK.
CK_TRUE	>1	Compute KEA $R_a$ value, store it in <i>pRandomA</i> , derive key value using full mode, create key object, return CKR_OK
CK_FALSE	0	Compute KEA $R_b$ value, store it in <i>pRandomB</i> , return CKR_OK. No derived key object is created.
CK_FALSE	1	Derive key value using e-mail mode, create key object, return CKR_OK.
CK_FALSE	>1	Derive key value using full mode, create key object, return CKR_OK.

Note that the parameter value *pRandomB* == 0 is a flag that the KEA mechanism is being invoked to compute the party's public random value ( $R_a$  or  $R_b$ , for sender or recipient, respectively), not to derive a

key. In these cases, any object template supplied as the **C\_DeriveKey** *pTemplate* argument should be ignored.

This mechanism has the following rules about key sensitivity and extractability\*:

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key MUST as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of KEA prime sizes, in bits.

## 2.4 RC2

### 2.4.1 Definitions

RC2 is a block cipher which is trademarked by RSA Security. It has a variable keysize and an additional parameter, the “effective number of bits in the RC2 search space”, which MAY take on values in the range 1-1024, inclusive. The effective number of bits in the RC2 search space is sometimes specified by an RC2 “version number”; this “version number” is *not* the same thing as the “effective number of bits”, however. There is a canonical way to convert from one to the other.

This section defines the key type “CKK\_RC2” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

CKM\_RC2\_KEY\_GEN  
CKM\_RC2\_ECB  
CKM\_RC2\_CBC  
CKM\_RC2\_MAC  
CKM\_RC2\_MAC\_GENERAL  
CKM\_RC2\_CBC\_PAD

### 2.4.2 RC2 secret key objects

RC2 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC2**) hold RC2 keys. The following table defines the RC2 secret key object attributes, in addition to the common attributes defined for this object class:

Table 6, RC2 Secret Key Object Attributes

Attribute	Data type	Meaning
-----------	-----------	---------

\* Note that the rules regarding the **CKA\_SENSITIVE**, **CKA\_EXTRACTABLE**, **CKA\_ALWAYS\_SENSITIVE**, and **CKA\_NEVER\_EXTRACTABLE** attributes have changed in version 2.11 to match the policy used by other key derivation mechanisms such as **CKM\_SSL3\_MASTER\_KEY\_DERIVE**.

CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (1 to 128 bytes)
CKA_VALUE_LEN <sup>2,3</sup>	CK_ULONG	Length in bytes of key value

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating an RC2 secret key object:

```

CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC2;
CK_UTF8CHAR label[] = "An RC2 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};

```

## 2.4.3 RC2 mechanism parameters

### 2.4.3.1 CK\_RC2\_PARAMS; CK\_RC2\_PARAMS\_PTR

**CK\_RC2\_PARAMS** provides the parameters to the **CKM\_RC2\_ECB** and **CKM\_RC2\_MAC** mechanisms. It holds the effective number of bits in the RC2 search space. It is defined as follows:

```
typedef CK_ULONG CK_RC2_PARAMS;
```

**CK\_RC2\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_PARAMS**.

### 2.4.3.2 CK\_RC2\_CBC\_PARAMS; CK\_RC2\_CBC\_PARAMS\_PTR

**CK\_RC2\_CBC\_PARAMS** is a structure that provides the parameters to the **CKM\_RC2\_CBC** and **CKM\_RC2\_CBC\_PAD** mechanisms. It is defined as follows:

```

typedef struct CK_RC2_CBC_PARAMS {
    CK_ULONG ulEffectiveBits;
    CK_BYTE iv[8];
} CK_RC2_CBC_PARAMS;

```

The fields of the structure have the following meanings:

*ulEffectiveBits*      the effective number of bits in the RC2 search space

*iv*      the initialization vector (IV) for cipher block chaining mode

**CK\_RC2\_CBC\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_CBC\_PARAMS**.

### 2.4.3.3 CK\_RC2\_MAC\_GENERAL\_PARAMS; CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR

**CK\_RC2\_MAC\_GENERAL\_PARAMS** is a structure that provides the parameters to the **CKM\_RC2\_MAC\_GENERAL** mechanism. It is defined as follows:

```

typedef struct CK_RC2_MAC_GENERAL_PARAMS {
    CK_ULONG ulEffectiveBits;
    CK_ULONG ulMacLength;
} CK_RC2_MAC_GENERAL_PARAMS;

```

429 The fields of the structure have the following meanings:

430 *ulEffectiveBits* the effective number of bits in the RC2 search space

431 *ulMacLength* length of the MAC produced, in bytes

432 **CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_MAC\_GENERAL\_PARAMS**.

#### 433 2.4.4 RC2 key generation

434 The RC2 key generation mechanism, denoted **CKM\_RC2\_KEY\_GEN**, is a key generation mechanism for  
 435 RSA Security's block cipher RC2.

436 It does not have a parameter.

437 The mechanism generates RC2 keys with a particular length in bytes, as specified in the  
 438 **CKA\_VALUE\_LEN** attribute of the template for the key.

439 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 440 key. Other attributes supported by the RC2 key type (specifically, the flags indicating which functions the  
 441 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

442 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 443 specify the supported range of RC2 key sizes, in bits.

#### 444 2.4.5 RC2-ECB

445 RC2-ECB, denoted **CKM\_RC2\_ECB**, is a mechanism for single- and multiple-part encryption and  
 446 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and electronic  
 447 codebook mode as defined in FIPS PUB 81.

448 It has a parameter, a **CK\_RC2\_PARAMS**, which indicates the effective number of bits in the RC2 search  
 449 space.

450 This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to  
 451 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
 452 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes  
 453 so that the resulting length is a multiple of eight. The output data is the same length as the padded input  
 454 data. It does not wrap the key type, key length, or any other information about the key; the application  
 455 must convey these separately.

456 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
 457 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
 458 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
 459 attribute of the new key; other attributes required by the key type must be specified in the template.

460 Constraints on key types and the length of data are summarized in the following table:

461 *Table 7 RC2-ECB: Key and Data Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	RC2	Multiple of 8	Same as input length	No final part
C_Decrypt	RC2	Multiple of 8	Same as input length	No final part
C_WrapKey	RC2	Any	Input length rounded up to multiple of 8	
C_UnwrapKey	RC2	Multiple of 8	Determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC2 effective number of bits.

## 2.4.6 RC2-CBC

RC2\_CBC, denoted **CKM\_RC2\_CBC**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and cipher-block chaining mode as defined in FIPS PUB 81.

It has a parameter, a **CK\_RC2\_CBC\_PARAMS** structure, where the first field indicates the effective number of bits in the RC2 search space, and the next field is the initialization vector for cipher block chaining mode.

This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes so that the resulting length is a multiple of eight. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key; the application must convey these separately.

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key type must be specified in the template.

Constraints on key types and the length of data are summarized in the following table:

Table 8, RC2-CBC: Key and Data Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	RC2	Multiple of 8	Same as input length	No final part
C_Decrypt	RC2	Multiple of 8	Same as input length	No final part
C_WrapKey	RC2	Any	Input length rounded up to multiple of 8	
C_UnwrapKey	RC2	Multiple of 8	Determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC2 effective number of bits.

## 2.4.7 RC2-CBC with PKCS padding

RC2-CBC with PKCS padding, denoted **CKM\_RC2\_CBC\_PAD**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2; cipher-block chaining mode as defined in FIPS PUB 81; and the block cipher padding method detailed in PKCS #7.

It has a parameter, a **CK\_RC2\_CBC\_PARAMS** structure, where the first field indicates the effective number of bits in the RC2 search space, and the next field is the initialization vector.

The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified for the **CKA\_VALUE\_LEN** attribute.

In addition to being able to wrap and unwrap secret keys, this mechanism MAY wrap and unwrap RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see **[PKCS #11-Curr]**, **Miscellaneous simple key derivation mechanisms** for details). The entries in the table below



498 for data length constraints when wrapping and unwrapping keys do not apply to wrapping and  
499 unwrapping private keys.

500 Constraints on key types and the length of data are summarized in the following table:

501 *Table 9, RC2-CBC with PKCS Padding: Key and Data Length*

Function	Key type	Input length	Output length
C_Encrypt	RC2	Any	Input length rounded up to multiple of 8
C_Decrypt	RC2	Multiple of 8	Between 1 and 8 bytes shorter than input length
C_WrapKey	RC2	Any	Input length rounded up to multiple of 8
C_UnwrapKey	RC2	Multiple of 8	Between 1 and 8 bytes shorter than input length

502 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
503 specify the supported range of RC2 effective number of bits.

## 504 2.4.8 General-length RC2-MAC

505 General-length RC2-MAC, denoted **CKM\_RC2\_MAC\_GENERAL**, is a mechanism for single-and  
506 multiple-part signatures and verification, based on RSA Security's block cipher RC2 and data  
507 authorization as defined in FIPS PUB 113.

508 It has a parameter, a **CK\_RC2\_MAC\_GENERAL\_PARAMS** structure, which specifies the effective  
509 number of bits in the RC2 search space and the output length desired from the mechanism.

510 The output bytes from this mechanism are taken from the start of the final RC2 cipher block produced in  
511 the MACing process.

512 Constraints on key types and the length of data are summarized in the following table:

513 *Table 10, General-length RC2-MAC: Key and Data Length*

Function	Key type	Data length	Signature length
C_Sign	RC2	Any	0-8, as specified in parameters
C_Verify	RC2	Any	0-8, as specified in parameters

514 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
515 specify the supported range of RC2 effective number of bits.

## 516 2.4.9 RC2-MAC

517 RC2-MAC, denoted by **CKM\_RC2\_MAC**, is a special case of the general-length RC2-MAC mechanism  
518 (see Section 2.4.8). Instead of taking a **CK\_RC2\_MAC\_GENERAL\_PARAMS** parameter, it takes a  
519 **CK\_RC2\_PARAMS** parameter, which only contains the effective number of bits in the RC2 search space.  
520 RC2-MAC produces and verifies 4-byte MACs.

521 Constraints on key types and the length of data are summarized in the following table:

522

523 *Table 11, RC2-MAC: Key and Data Length*

Function	Key type	Data length	Signature length
C_Sign	RC2	Any	4
C_Verify	RC2	Any	4

524 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
525 specify the supported range of RC2 effective number of bits.

## 2.5 RC4

### 2.5.1 Definitions

This section defines the key type “CKK\_RC4” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms

CKM\_RC4\_KEY\_GEN

CKM\_RC4

### 2.5.2 RC4 secret key objects

RC4 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC4**) hold RC4 keys. The following table defines the RC4 secret key object attributes, in addition to the common attributes defined for this object class:

Table 12, RC4 Secret Key Object

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (1 to 256 bytes)
CKA_VALUE_LEN <sup>2,3,6</sup>	CK_ULONG	Length in bytes of key value

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating an RC4 secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC4;
CK_UTF8CHAR label[] = "An RC4 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

### 2.5.3 RC4 key generation

The RC4 key generation mechanism, denoted **CKM\_RC4\_KEY\_GEN**, is a key generation mechanism for RSA Security's proprietary stream cipher RC4.

It does not have a parameter.

The mechanism generates RC4 keys with a particular length in bytes, as specified in the **CKA\_VALUE\_LEN** attribute of the template for the key.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new key. Other attributes supported by the RC4 key type (specifically, the flags indicating which functions the key supports) MAY be specified in the template for the key, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC4 key sizes, in bits.



## 2.5.4 RC4 mechanism

RC4, denoted **CKM\_RC4**, is a mechanism for single- and multiple-part encryption and decryption based on RSA Security's proprietary stream cipher RC4.

It does not have a parameter.

Constraints on key types and the length of input and output data are summarized in the following table:

Table 13, RC4: Key and Data Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	RC4	Any	Same as input length	No final part
C_Decrypt	RC4	Any	Same as input length	No final part

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC4 key sizes, in bits.

## 2.6 RC5

### 2.6.1 Definitions

RC5 is a parameterizable block cipher patented by RSA Security. It has a variable wordsize, a variable keysize, and a variable number of rounds. The blocksize of RC5 is equal to twice its wordsize.

This section defines the key type "CKK\_RC5" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

- CKM\_RC5\_KEY\_GEN
- CKM\_RC5\_ECB
- CKM\_RC5\_CBC
- CKM\_RC5\_MAC
- CKM\_RC5\_MAC\_GENERAL
- CMK\_RC5\_CBC\_PAD

### 2.6.2 RC5 secret key objects

RC5 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC5**) hold RC5 keys. The following table defines the RC5 secret key object attributes, in addition to the common attributes defined for this object class.

Table 14, RC5 Secret Key Object

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (0 to 255 bytes)
CKA_VALUE_LEN <sup>2,3,6</sup>	CK_ULONG	Length in bytes of key value

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating an RC5 secret key object:

```
CK OBJECT CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC5;
CK_UTF8CHAR label[] = "An RC5 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
```

```

598 CK_ATTRIBUTE template[] = {
599     {CKA_CLASS, &class, sizeof(class)},
600     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
601     {CKA_TOKEN, &true, sizeof(true)},
602     {CKA_LABEL, label, sizeof(label)-1},
603     {CKA_ENCRYPT, &true, sizeof(true)},
604     {CKA_VALUE, value, sizeof(value)}
605 };

```

## 606 2.6.3 RC5 mechanism parameters

### 607 2.6.3.1 CK\_RC5\_PARAMS; CK\_RC5\_PARAMS\_PTR

608 **CK\_RC5\_PARAMS** provides the parameters to the **CKM\_RC5\_ECB** and **CKM\_RC5\_MAC** mechanisms.  
609 It is defined as follows:

```

610 typedef struct CK_RC5_PARAMS {
611     CK_ULONG ulWordsize;
612     CK_ULONG ulRounds;
613 } CK_RC5_PARAMS;

```

614 The fields of the structure have the following meanings:

615 *ulWordsize*      wordsize of RC5 cipher in bytes

616 *ulRounds*      number of rounds of RC5 encipherment

617 **CK\_RC5\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_PARAMS**.

### 618 2.6.3.2 CK\_RC5\_CBC\_PARAMS; CK\_RC5\_CBC\_PARAMS\_PTR

619 **CK\_RC5\_CBC\_PARAMS** is a structure that provides the parameters to the **CKM\_RC5\_CBC** and  
620 **CKM\_RC5\_CBC\_PAD** mechanisms. It is defined as follows:

```

621 typedef struct CK_RC5_CBC_PARAMS {
622     CK_ULONG ulWordsize;
623     CK_ULONG ulRounds;
624     CK_BYTE_PTR pIv;
625     CK_ULONG ulIvLen;
626 } CK_RC5_CBC_PARAMS;

```

627 The fields of the structure have the following meanings:

628 *ulwordSize*      wordsize of RC5 cipher in bytes

629 *ulRounds*      number of rounds of RC5 encipherment

630 *pIv*      pointer to initialization vector (IV) for CBC encryption

631 *ulIvLen*      length of initialization vector (must be same as  
632                      blocksize)

633 **CK\_RC5\_CBC\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_CBC\_PARAMS**.

### 634 2.6.3.3 CK\_RC5\_MAC\_GENERAL\_PARAMS; 635 CK\_RC5\_MAC\_GENERAL\_PARAMS\_PTR

636 **CK\_RC5\_MAC\_GENERAL\_PARAMS** is a structure that provides the parameters to the  
637 **CKM\_RC5\_MAC\_GENERAL** mechanism. It is defined as follows:

```

638 typedef struct CK_RC5_MAC_GENERAL_PARAMS {
639     CK_ULONG ulWordsize;
640     CK_ULONG ulRounds;
641     CK_ULONG ulMacLength;
642 } CK_RC5_MAC_GENERAL_PARAMS;

```

643 The fields of the structure have the following meanings:

644                   *ulwordSize*      wordsize of RC5 cipher in bytes

645                   *ulRounds*        number of rounds of RC5 encipherment

646                   *ulMacLength*    length of the MAC produced, in bytes

647 **CK\_RC5\_MAC\_GENERAL\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_MAC\_GENERAL\_PARAMS**.

## 648 2.6.4 RC5 key generation

649 The RC5 key generation mechanism, denoted **CKM\_RC5\_KEY\_GEN**, is a key generation mechanism for  
650 RSA Security's block cipher RC5.

651 It does not have a parameter.

652 The mechanism generates RC5 keys with a particular length in bytes, as specified in the  
653 **CKA\_VALUE\_LEN** attribute of the template for the key.

654 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
655 key. Other attributes supported by the RC5 key type (specifically, the flags indicating which functions the  
656 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

657 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
658 specify the supported range of RC5 key sizes, in bytes.

## 659 2.6.5 RC5-ECB

660 RC5-ECB, denoted **CKM\_RC5\_ECB**, is a mechanism for single- and multiple-part encryption and  
661 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and electronic  
662 codebook mode as defined in FIPS PUB 81.

663 It has a parameter, **CK\_RC5\_PARAMS**, which indicates the wordsize and number of rounds of  
664 encryption to use.

665 This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to  
666 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
667 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with null bytes so that the  
668 resulting length is a multiple of the cipher blocksize (twice the wordsize). The output data is the same  
669 length as the padded input data. It does not wrap the key type, key length, or any other information about  
670 the key; the application must convey these separately.

671 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
672 **CKA\_KEY\_TYPE** attributes of the template and, if it has one, and the key type supports it, the  
673 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
674 attribute of the new key; other attributes required by the key type must be specified in the template.

675 Constraints on key types and the length of data are summarized in the following table:

676 *Table 15, RC5-ECB Key and Data Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	RC5	Multiple of blocksize	Same as input length	No final part

C_Decrypt	RC5	Multiple of blocksize	Same as input length	No final part
C_WrapKey	RC5	Any	Input length rounded up to multiple of blocksize	
C_UnwrapKey	RC5	Multiple of blocksize	Determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC5 key sizes, in bytes.

## 2.6.6 RC5-CBC

RC5-CBC, denoted **CKM\_RC5\_CBC**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and cipher-block chaining mode as defined in FIPS PUB 81.

It has a parameter, a **CK\_RC5\_CBC\_PARAMS** structure, which specifies the wordsize and number of rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.

This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes so that the resulting length is a multiple of eight. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key; the application must convey these separately.

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA\_KEY\_TYPE** attribute for the template, and, if it has one, and the key type supports it, the **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key type must be specified in the template.

Constraints on key types and the length of data are summarized in the following table:

*Table 16, RC5-CBC Key and Data Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	RC5	Multiple of blocksize	Same as input length	No final part
C_Decrypt	RC5	Multiple of blocksize	Same as input length	No final part
C_WrapKey	RC5	Any	Input length rounded up to multiple of blocksize	
C_UnwrapKey	RC5	Multiple of blocksize	Determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC5 key sizes, in bytes.

## 2.6.7 RC5-CBC with PKCS padding

RC5-CBC with PKCS padding, denoted **CKM\_RC5\_CBC\_PAD**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5; cipher block chaining mode as defined in FIPS PUB 81; and the block cipher padding method detailed in PKCS #7.

It has a parameter, a **CK\_RC5\_CBC\_PARAMS** structure, which specifies the wordsize and number of rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.

The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified for the **CKA\_VALUE\_LEN** attribute.

In addition to being able to wrap and unwrap secret keys, this mechanism MAY wrap and unwrap RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys. The entries in the table below for data length constraints when wrapping and unwrapping keys do not apply to wrapping and unwrapping private keys.

Constraints on key types and the length of data are summarized in the following table:

Table 17, RC5-CBC with PKCS Padding; Key and Data Length

Function	Key type	Input length	Output length
C_Encrypt	RC5	Any	Input length rounded up to multiple of blocksize
C_Decrypt	RC5	Multiple of blocksize	Between 1 and blocksize bytes shorter than input length
C_WrapKey	RC5	Any	Input length rounded up to multiple of blocksize
C_UnwrapKey	RC5	Multiple of blocksize	Between 1 and blocksize bytes shorter than input length

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC5 key sizes, in bytes.

## 2.6.8 General-length RC5-MAC

General-length RC5-MAC, denoted **CKM\_RC5\_MAC\_GENERAL**, is a mechanism for single- and multiple-part signatures and verification, based on RSA Security's block cipher RC5 and data authentication as defined in FIPS PUB 113.

It has a parameter, a **CK\_RC5\_MAC\_GENERAL\_PARAMS** structure, which specifies the wordsize and number of rounds of encryption to use and the output length desired from the mechanism.

The output bytes from this mechanism are taken from the start of the final RC5 cipher block produced in the MACing process.

Constraints on key types and the length of data are summarized in the following table:

Table 18, General-length RC2-MAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	RC5	Any	0-blocksize, as specified in parameters
C_Verify	RC5	Any	0-blocksize, as specified in parameters

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC5 key sizes, in bytes.

## 2.6.9 RC5-MAC

RC5-MAC, denoted by **CKM\_RC5\_MAC**, is a special case of the general-length RC5-MAC mechanism. Instead of taking a **CK\_RC5\_MAC\_GENERAL\_PARAMS** parameter, it takes a **CK\_RC5\_PARAMS** parameter. RC5-MAC produces and verifies MACs half as large as the RC5 blocksize.

Constraints on key types and the length of data are summarized in the following table:

Table 19, RC5-MAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	RC5	Any	RC5 wordsize = [blocksize/2]
C_Verify	RC5	Any	RC5 wordsize = [blocksize/2]

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of RC5 key sizes, in bytes.

## 2.7 General block cipher

### 2.7.1 Definitions

For brevity's sake, the mechanisms for the DES, CAST, CAST3, CAST128 (CAST5), IDEA and CDMF block ciphers are described together here. Each of these ciphers has the following mechanisms, which are described in a templated form.

This section defines the key types "CKK\_DES", "CKK\_CAST", "CKK\_CAST3", "CKK\_CAST5" (deprecated in v2.11), "CKK\_CAST128", "CKK\_IDEA" and "CKK\_CDMF" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

- CKM\_DES\_KEY\_GEN
- CKM\_DES\_ECB
- CKM\_DES\_CBC
- CKM\_DES\_MAC
- CKM\_DES\_MAC\_GENERAL
- CKM\_DES\_CBC\_PAD
- CKM\_CDMF\_KEY\_GEN
- CKM\_CDMF\_ECB
- CKM\_CDMF\_CBC
- CKM\_CDMF\_MAC
- CKM\_CDMF\_MAC\_GENERAL
- CKM\_CDMF\_CBC\_PAD
- CKM\_DES\_OFB64
- CKM\_DES\_OFB8
- CKM\_DES\_CFB64
- CKM\_DES\_CFB8
- CKM\_CAST\_KEY\_GEN
- CKM\_CAST\_ECB
- CKM\_CAST\_CBC
- CKM\_CAST\_MAC
- CKM\_CAST\_MAC\_GENERAL
- CKM\_CAST\_CBC\_PAD
- CKM\_CAST3\_KEY\_GEN
- CKM\_CAST3\_ECB
- CKM\_CAST3\_CBC
- CKM\_CAST3\_MAC
- CKM\_CAST3\_MAC\_GENERAL

773 CKM\_CAST3\_CBC\_PAD  
 774 CKM\_CAST5\_KEY\_GEN  
 775 CKM\_CAST128\_KEY\_GEN  
 776 CKM\_CAST5\_ECB  
 777 CKM\_CAST128\_ECB  
 778 CKM\_CAST5\_CBC  
 779 CKM\_CAST128\_CBC  
 780 CKM\_CAST5\_MAC  
 781 CKM\_CAST128\_MAC  
 782 CKM\_CAST5\_MAC\_GENERAL  
 783 CKM\_CAST128\_MAC\_GENERAL  
 784 CKM\_CAST5\_CBC\_PAD  
 785 CKM\_CAST128\_CBC\_PAD  
 786 CKM\_IDEA\_KEY\_GEN  
 787 CKM\_IDEA\_ECB  
 788 CKM\_IDEA\_MAC  
 789 CKM\_IDEA\_MAC\_GENERAL  
 790 CKM\_IDEA\_CBC\_PAD

## 791 2.7.2 DES secret key objects

792 DES secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_DES**) hold single-length DES  
 793 keys. The following table defines the DES secret key object attributes, in addition to the common  
 794 attributes defined for this object class:

795 *Table 20, DES Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (8 bytes long)

796 Refer to [PKCS #11-Base] table 15 for footnotes

797 DES keys MUST have their parity bits properly set as described in FIPS PUB 46-3. Attempting to create  
 798 or unwrap a DES key with incorrect parity MUST return an error.

799 The following is a sample template for creating a DES secret key object:

```

800 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
801 CK_KEY_TYPE keyType = CKK_DES;
802 CK_UTF8CHAR label[] = "A DES secret key object";
803 CK_BYTE value[8] = {...};
804 CK_BBOOL true = CK_TRUE;
805 CK_ATTRIBUTE template[] = {
806   {CKA_CLASS, &class, sizeof(class)},
807   {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
808   {CKA_TOKEN, &true, sizeof(true)},
809   {CKA_LABEL, label, sizeof(label)-1},
810   {CKA_ENCRYPT, &true, sizeof(true)},
811   {CKA_VALUE, value, sizeof(value)}
812 };
  
```

813 CKA\_CHECK\_VALUE: The value of this attribute is derived from the key object by taking the first three  
 814 bytes of the ECB encryption of a single block of null (0x00) bytes, using the default cipher associated with  
 815 the key type of the secret key object.



## 2.7.3 CAST secret key objects

CAST secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_CAST**) hold CAST keys. The following table defines the CAST secret key object attributes, in addition to the common attributes defined for this object class:

Table 21, CAST Secret Key Object Attributes

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (1 to 8 bytes)
CKA_VALUE_LEN <sup>2,3,6</sup>	CK_ULONG	Length in bytes of key value

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating a CAST secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST;
CK_UTF8CHAR label[] = "A CAST secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.7.4 CAST3 secret key objects

CAST3 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_CAST3**) hold CAST3 keys. The following table defines the CAST3 secret key object attributes, in addition to the common attributes defines for this object class:

Table 22, CAST3 Secret Key Object Attributes

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (1 to 8 bytes)
CKA_VALUE_LEN <sup>2,3,6</sup>	CK_ULONG	Length in bytes of key value

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating a CAST3 secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST3;
CK_UTF8CHAR label[] = "A CAST3 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```



## 2.7.5 CAST128 (CAST5) secret key objects

CAST128 (also known as CAST5) secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_CAST128** or **CKK\_CAST5**) hold CAST128 keys. The following table defines the CAST128 secret key object attributes, in addition to the common attributes defines for this object class:

Table 23, CAST128 (CAST5) Secret Key Object Attributes

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (1 to 16 bytes)
CKA_VALUE_LEN <sup>2,3,6</sup>	CK_ULONG	Length in bytes of key value

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating a CAST128 (CAST5) secret key object:

```
CK OBJECT CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST128;
CK_UTF8CHAR label[] = "A CAST128 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.7.6 IDEA secret key objects

IDEA secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_IDEA**) hold IDEA keys. The following table defines the IDEA secret key object attributes, in addition to the common attributes defines for this object class:

Table 24, IDEA Secret Key Object

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (16 bytes long)

Refer to [PKCS #11-Base] table 15 for footnotes

The following is a sample template for creating an IDEA secret key object:

```
CK OBJECT CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_IDEA;
CK_UTF8CHAR label[] = "An IDEA secret key object";
CK_BYTE value[16] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.7.7 CDMF secret key objects

IDEA secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_CDMF**) hold CDMF keys. The following table defines the CDMF secret key object attributes, in addition to the common attributes defines for this object class:

Table 25, CDMF Secret Key Object

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (8 bytes long)

Refer to [PKCS #11-Base] table 15 for footnotes

CDMF keys MUST have their parity bits properly set in exactly the same fashion described for DES keys in FIPS PUB 46-3. Attempting to create or unwrap a CDMF key with incorrect parity MUST return an error.

The following is a sample template for creating a CDMF secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CDMF;
CK_UTF8CHAR label[] = "A CDMF secret key object";
CK_BYTE value[8] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.7.8 General block cipher mechanism parameters

### 2.7.8.1 CK\_MAC\_GENERAL\_PARAMS; CK\_MAC\_GENERAL\_PARAMS\_PTR

**CK\_MAC\_GENERAL\_PARAMS** provides the parameters to the general-length MACing mechanisms of the DES, DES3 (triple-DES), CAST, CAST3, CAST128 (CAST5), IDEA, CDMF and AES ciphers. It also provides the parameters to the general-length HMACing mechanisms (i.e., MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-128 and RIPEMD-160) and the two SSL 3.0 MACing mechanisms, (i.e., MD5 and SHA-1). It holds the length of the MAC that these mechanisms produce. It is defined as follows:

```
typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
```

**CK\_MAC\_GENERAL\_PARAMS\_PTR** is a pointer to a **CK\_MAC\_GENERAL\_PARAMS**.

## 2.7.9 General block cipher key generation

Cipher <NAME> has a key generation mechanism, "<NAME> key generation", denoted by **CKM\_<NAME>\_KEY\_GEN**.

This mechanism does not have a parameter.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new key. Other attributes supported by the key type (specifically, the flags indicating which functions the key supports) MAY be specified in the template for the key, or else are assigned default initial values.

When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits set properly.

When DES or CDMF keys are generated, it is token-dependent whether or not it is possible for “weak” or “semi-weak” keys to be generated. Similarly, when triple-DES keys are generated, it is token-dependent whether or not it is possible for any of the component DES keys to be “weak” or “semi-weak” keys.

When CAST, CAST3, or CAST128 (CAST5) keys are generated, the template for the secret key must specify a **CKA\_VALUE\_LEN** attribute.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for the key generation mechanisms for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

## 2.7.10 General block cipher ECB

Cipher <NAME> has an electronic codebook mechanism, “<NAME>-ECB”, denoted **CKM\_<NAME>\_ECB**. It is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping with <NAME>.

It does not have a parameter.

This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with null bytes so that the resulting length is a multiple of <NAME>’s blocksize. The output data is the same length as the padded input data. It does not wrap the key type, key length or any other information about the key; the application must convey these separately.

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key must be specified in the template.

Constraints on key types and the length of data are summarized in the following table:

Table 26, General Block Cipher ECB: Key and Data Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	<NAME>	Multiple of blocksize	Same as input length	No final part
C_Decrypt	<NAME>	Multiple of blocksize	Same as input length	No final part
C_WrapKey	<NAME>	Any	Input length rounded up to multiple of blocksize	
C_UnwrapKey	<NAME>	Any	Determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

## 2.7.11 General block cipher CBC

Cipher <NAME> has a cipher-block chaining mode, “<NAME>-CBC”, denoted **CKM\_<NAME>\_CBC**. It is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping with <NAME>.

977 It has a parameter, an initialization vector for cipher block chaining mode. The initialization vector has the  
 978 same length as <NAME>'s blocksize.

979 Constraints on key types and the length of data are summarized in the following table:

980 *Table 27, General Block Cipher CBC; Key and Data Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	<NAME>	Multiple of blocksize	Same as input length	No final part
C_Decrypt	<NAME>	Multiple of blocksize	Same as input length	No final part
C_WrapKey	<NAME>	Any	Input length rounded up to multiple of blocksize	
C_UnwrapKey	<NAME>	Any	Determined by type of key being unwrapped or CKA_VALUE_LEN	

981 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 982 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for  
 983 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 984 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA, and CDMF  
 985 ciphers, these fields are not used.

## 986 2.7.12 General block cipher CBC with PKCS padding

987 Cipher <NAME> has a cipher-block chaining mode with PKCS padding, "<NAME>-CBC with PKCS  
 988 padding", denoted **CKM\_<NAME>\_CBC\_PAD**. It is a mechanism for single- and multiple-part encryption  
 989 and decryption; key wrapping; and key unwrapping with <NAME>. All ciphertext is padded with PKCS  
 990 padding.

991 It has a parameter, an initialization vector for cipher block chaining mode. The initialization vector has the  
 992 same length as <NAME>'s blocksize.

993 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
 994 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
 995 for the **CKA\_VALUE\_LEN** attribute.

996

997 In addition to being able to wrap and unwrap secret keys, this mechanism MAY wrap and unwrap RSA,  
 998 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys. The entries in  
 999 the table below for data length constraints when wrapping and unwrapping keys to not apply to wrapping  
 1000 and unwrapping private keys.

1001 Constraints on key types and the length of data are summarized in the following table:

1002 *Table 28, General Block Cipher CBC with PKCS Padding: Key and Data Length*

Function	Key type	Input length	Output length
C_Encrypt	<NAME>	Any	Input length rounded up to multiple of blocksize
C_Decrypt	<NAME>	Multiple of blocksize	Between 1 and blocksize bytes shorter than input length
C_WrapKey	<NAME>	Any	Input length rounded up to multiple of blocksize
C_UnwrapKey	<NAME>	Multiple of	Between 1 and blocksize bytes shorter than input

		blocksize	length
--	--	-----------	--------

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure MAY be used. The CAST, CAST3 and CAST128 (CAST5) ciphers have variable key sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA, and CDMF ciphers, these fields are not used.

### 2.7.13 General-length general block cipher MAC

Cipher <NAME> has a general-length MACing mode, "General-length <NAME>-MAC", denoted **CKM\_<NAME>\_MAC\_GENERAL**. It is a mechanism for single-and multiple-part signatures and verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB 113.

It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which specifies the size of the output.

The output bytes from this mechanism are taken from the start of the final cipher block produced in the MACing process.

Constraints on key types and the length of input and output data are summarized in the following table:

Table 29, General-length General Block Cipher MAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	<NAME>	Any	0-blocksize, depending on parameters
C_Verify	<NAME>	Any	0-blocksize, depending on parameters

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

### 2.7.14 General block cipher MAC

Cipher <NAME> has a MACing mechanism, "<NAME>-MAC", denoted **CKM\_<NAME>\_MAC**. This mechanism is a special case of the **CKM\_<NAME>\_MAC\_GENERAL** mechanism described above. It produces an output of size half as large as <NAME>'s blocksize.

This mechanism has no parameters.

Constraints on key types and the length of data are summarized in the following table:

Table 30, General Block Cipher MAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	<NAME>	Any	[blocksize/2]
C_Verify	<NAME>	Any	[blocksize/2]

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

## 2.8 SKIPJACK

### 2.8.1 Definitions

This section defines the key type “CKK\_SKIPJACK” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

- CKM\_SKIPJACK\_KEY\_GEN
- CKM\_SKIPJACK\_ECB64
- CKM\_SKIPJACK\_CBC64
- CKM\_SKIPJACK\_OFB64
- CKM\_SKIPJACK\_CFB64
- CKM\_SKIPJACK\_CFB32
- CKM\_SKIPJACK\_CFB16
- CKM\_SKIPJACK\_CFB8
- CKM\_SKIPJACK\_WRAP
- CKM\_SKIPJACK\_PRIVATE\_WRAP
- CKM\_SKIPJACK\_RELAYX

### 2.8.2 SKIPJACK secret key objects

SKIPJACK secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_SKIPJACK**) holds a single-length MEK or a TEK. The following table defines the SKIPJACK secret object attributes, in addition to the common attributes defined for this object class:

Table 31, SKIPJACK Secret Key Object

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (12 bytes long)

Refer to [PKCS #11-Base] table 15 for footnotes

SKIPJACK keys have 16 checksum bits, and these bits must be properly set. Attempting to create or unwrap a SKIPJACK key with incorrect checksum bits MUST return an error.

It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK key with a specified value. Nonetheless, we provide templates for doing so.

The following is a sample template for creating a SKIPJACK MEK secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SKIPJACK;
CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";
CK_BYTE value[12] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

The following is a sample template for creating a SKIPJACK TEK secret key object:



```

1077 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1078 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1079 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1080 CK_BYTE value[12] = {...};
1081 CK_BBOOL true = CK_TRUE;
1082 CK_ATTRIBUTE template[] = {
1083     {CKA_CLASS, &class, sizeof(class)},
1084     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1085     {CKA_TOKEN, &true, sizeof(true)},
1086     {CKA_LABEL, label, sizeof(label)-1},
1087     {CKA_ENCRYPT, &true, sizeof(true)},
1088     {CKA_WRAP, &true, sizeof(true)},
1089     {CKA_VALUE, value, sizeof(value)}
1090 };

```

## 2.8.3 SKIPJACK Mechanism parameters

### 2.8.3.1 CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS; CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR

**CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS** is a structure that provides the parameters to the **CKM\_SKIPJACK\_PRIVATE\_WRAP** mechanism. It is defined as follows:

```

1096 typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
1097     CK_ULONG ulPasswordLen;
1098     CK_BYTE_PTR pPassword;
1099     CK_ULONG ulPublicDataLen;
1100     CK_BYTE_PTR pPublicData;
1101     CK_ULONG ulPandGLen;
1102     CK_ULONG ulQLen;
1103     CK_ULONG ulRandomLen;
1104     CK_BYTE_PTR pRandomA;
1105     CK_BYTE_PTR pPrimeP;
1106     CK_BYTE_PTR pBaseG;
1107     CK_BYTE_PTR pSubprimeQ;
1108 } CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

```

The fields of the structure have the following meanings:

<i>ulPasswordLen</i>	length of the password
<i>pPassword</i>	pointer to the buffer which contains the user-supplied password
<i>ulPublicDataLen</i>	other party's key exchange public key size
<i>pPublicData</i>	pointer to other party's key exchange public key value
<i>ulPandGLen</i>	length of prime and base values
<i>ulQLen</i>	length of subprime value
<i>ulRandomLen</i>	size of random Ra, in bytes
<i>pPrimeP</i>	pointer to Prime, p, value
<i>pBaseG</i>	pointer to Base, b, value

1120                    *pSubprimeQ*      pointer to Subprime, q, value

1121    **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR** is a pointer to a  
1122                    **CK\_PRIVATE\_WRAP\_PARAMS**.

1123    **2.8.3.2 CK\_SKIPJACK\_RELAYX\_PARAMS;**  
1124                    **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR**

1125    **CK\_SKIPJACK\_RELAYX\_PARAMS** is a structure that provides the parameters to the  
1126    **CKM\_SKIPJACK\_RELAYX** mechanism. It is defined as follows:

```
1127    typedef struct CK_SKIPJACK_RELAYX_PARAMS {  
1128       CK_ULONG ulOldWrappedXLen;  
1129       CK_BYTE_PTR pOldWrappedX;  
1130       CK_ULONG ulOldPasswordLen;  
1131       CK_BYTE_PTR pOldPassword;  
1132       CK_ULONG ulOldPublicDataLen;  
1133       CK_BYTE_PTR pOldPublicData;  
1134       CK_ULONG ulOldRandomLen;  
1135       CK_BYTE_PTR pOldRandomA;  
1136       CK_ULONG ulNewPasswordLen;  
1137       CK_BYTE_PTR pNewPassword;  
1138       CK_ULONG ulNewPublicDataLen;  
1139       CK_BYTE_PTR pNewPublicData;  
1140       CK_ULONG ulNewRandomLen;  
1141       CK_BYTE_PTR pNewRandomA;  
1142    } CK_SKIPJACK_RELAYX_PARAMS;
```

1143    The fields of the structure have the following meanings:

1144                    *ulOldWrappedLen*      length of old wrapped key in bytes

1145                    *pOldWrappedX*       pointer to old wrapper key

1146                    *ulOldPasswordLen*    length of the old password

1147                    *pOldPassword*       pointer to the buffer which contains the old user-supplied  
1148                    password

1149                    *ulOldPublicDataLen*    old key exchange public key size

1150                    *pOldPublicData*       pointer to old key exchange public key value

1151                    *ulOldRandomLen*       size of old random Ra in bytes

1152                    *pOldRandomA*       pointer to old Ra data

1153                    *ulNewPasswordLen*    length of the new password

1154                    *pNewPassword*       pointer to the buffer which contains the new user-  
1155                    supplied password

1156                    *ulNewPublicDataLen*    new key exchange public key size

1157                    *pNewPublicData*       pointer to new key exchange public key value



1158 *ulNewRandomLen* size of new random Ra in bytes

1159 *pNewRandomA* pointer to new Ra data

1160 **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR** is a pointer to a **CK\_SKIPJACK\_RELAYX\_PARAMS**.

## 1161 2.8.4 SKIPJACK key generation

1162 The SKIPJACK key generation mechanism, denoted **CKM\_SKIPJACK\_KEY\_GEN**, is a key generation  
1163 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1164 It does not have a parameter.

1165 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1166 key.

## 1167 2.8.5 SKIPJACK-ECB64

1168 SKIPJACK-ECB64, denoted **CKM\_SKIPJACK\_ECB64**, is a mechanism for single- and multiple-part  
1169 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB  
1170 185.

1171 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1172 value generated by the token – in other words, the application cant specify a particular IV when  
1173 encrypting. It MAY, of course, specify a particular IV when decrypting.

1174 Constraints on key types and the length of data are summarized in the following table:

1175 *Table 32, SKIPJACK-ECB64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

## 1176 2.8.6 SKIPJACK-CBC64

1177 SKIPJACK-CBC64, denoted **CKM\_SKIPJACK\_CBC64**, is a mechanism for single- and multiple-part  
1178 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1179 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1180 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1181 encrypting. It MAY, of course, specify a particular IV when decrypting.

1182 Constraints on key types and the length of data are summarized in the following table:

1183 *Table 33, SKIPJACK-CBC64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

## 1184 2.8.7 SKIPJACK-OFB64

1185 SKIPJACK-OFB64, denoted **CKM\_SKIPJACK\_OFB64**, is a mechanism for single- and multiple-part  
1186 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1187 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1188 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1189 encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

*Table 34, SKIPJACK-OFB64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

## 2.8.8 SKIPJACK-CFB64

SKIPJACK-CFB64, denoted **CKM\_SKIPJACK\_CFB64**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

*Table 35, SKIPJACK-CFB64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

## 2.8.9 SKIPJACK-CFB32

SKIPJACK-CFB32, denoted **CKM\_SKIPJACK\_CFB32**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

*Table 36, SKIPJACK-CFB32: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 4	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 4	Same as input length	No final part

## 2.8.10 SKIPJACK-CFB16

SKIPJACK-CFB16, denoted **CKM\_SKIPJACK\_CFB16**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

*Table 37, SKIPJACK-CFB16: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 4	Same as input length	No final part

C_Decrypt	SKIPJACK	Multiple of 4	Same as input length	No final part
-----------	----------	---------------	----------------------	---------------

## 2.8.11 SKIPJACK-CFB8

SKIPJACK-CFB8, denoted **CKM\_SKIPJACK\_CFB8**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

*Table 38, SKIPJACK-CFB8: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 4	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 4	Same as input length	No final part

## 2.8.12 SKIPJACK-WRAP

The SKIPJACK-WRAP mechanism, denoted **CKM\_SKIPJACK\_WRAP**, is used to wrap and unwrap a secret key (MEK). It MAY wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.

It does not have a parameter.

## 2.8.13 SKIPJACK-PRIVATE-WRAP

The SKIPJACK-PRIVATE-WRAP mechanism, denoted **CKM\_SKIPJACK\_PRIVATE\_WRAP**, is used to wrap and unwrap a private key. It MAY wrap KEA and DSA private keys.

It has a parameter, a **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS** structure.

## 2.8.14 SKIPJACK-RELAYX

The SKIPJACK-RELAYX mechanism, denoted **CKM\_SKIPJACK\_RELAYX**, is used with the **C\_WrapKey** function to “change the wrapping” on a private key which was wrapped with the SKIPJACK-PRIVATE-WRAP mechanism (See Section 2.8.13).

It has a parameter, a **CK\_SKIPJACK\_RELAYX\_PARAMS** structure.

Although the SKIPJACK-RELAYX mechanism is used with **C\_WrapKey**, it differs from other key-wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to **C\_WrapKey**; however for the SKIPJACK\_RELAYX mechanism, the [always invalid] value 0 should be passed as the key handle for **C\_WrapKey**, and the already-wrapped key should be passed in as part of the **CK\_SKIPJACK\_RELAYX\_PARAMS** structure.

# 2.9 BATON

## 2.9.1 Definitions

This section defines the key type “CKK\_BATON” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

CKM\_BATON\_KEY\_GEN

CKM\_BATON\_ECB128

CKM\_BATON\_ECB96

- 1250 CKM\_BATON\_CBC128
- 1251 CKM\_BATON\_COUNTER
- 1252 CKM\_BATON\_SHUFFLE
- 1253 CKM\_BATON\_WRAP

1254 **2.9.2 BATON secret key objects**

1255 BATON secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_BATON**) hold single-length  
 1256 BATON keys. The following table defines the BATON secret key object attributes, in addition to the  
 1257 common attributes defined for this object class:

1258 *Table 39, BATON Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (40 bytes long)

1259 Refer to [PKCS #11-Base] table 15 for footnotes

1260

1261 BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or  
 1262 unwrap a BATON key with incorrect checksum bits MUST return an error.

1263 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key  
 1264 with a specified value. Nonetheless, we provide templates for doing so.

1265 The following is a sample template for creating a BATON MEK secret key object:

```

1266 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1267 CK_KEY_TYPE keyType = CKK_BATON;
1268 CK_UTF8CHAR label[] = "A BATON MEK secret key object";
1269 CK_BYTE value[40] = {...};
1270 CK_BBOOL true = CK_TRUE;
1271 CK_ATTRIBUTE template[] = {
1272     {CKA_CLASS, &class, sizeof(class)},
1273     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1274     {CKA_TOKEN, &true, sizeof(true)},
1275     {CKA_LABEL, label, sizeof(label)-1},
1276     {CKA_ENCRYPT, &true, sizeof(true)},
1277     {CKA_VALUE, value, sizeof(value)}
1278 };
  
```

1279 The following is a sample template for creating a BATON TEK secret key object:

```

1280 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1281 CK_KEY_TYPE keyType = CKK_BATON;
1282 CK_UTF8CHAR label[] = "A BATON TEK secret key object";
1283 CK_BYTE value[40] = {...};
1284 CK_BBOOL true = CK_TRUE;
1285 CK_ATTRIBUTE template[] = {
1286     {CKA_CLASS, &class, sizeof(class)},
1287     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1288     {CKA_TOKEN, &true, sizeof(true)},
1289     {CKA_LABEL, label, sizeof(label)-1},
1290     {CKA_ENCRYPT, &true, sizeof(true)},
1291     {CKA_WRAP, &true, sizeof(true)},
1292     {CKA_VALUE, value, sizeof(value)}
1293 };
  
```

1294 **2.9.3 BATON key generation**

1295 The BATON key generation mechanism, denoted **CKM\_BATON\_KEY\_GEN**, is a key generation  
 1296 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

It does not have a parameter.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new key.

## 2.9.4 BATON-ECB128

BATON-ECB128, denoted **CKM\_BATON\_ECB128**, is a mechanism for single- and multiple-part encryption and decryption with BATON in 128-bit electronic codebook mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

Table 40, BATON-ECB128: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	BATON	Multiple of 16	Same as input length	No final part
C_Decrypt	BATON	Multiple of 16	Same as input length	No final part

## 2.9.5 BATON-ECB96

BATON-ECB96, denoted **CKM\_BATON\_ECB96**, is a mechanism for single- and multiple-part encryption and decryption with BATON in 96-bit electronic codebook mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

Table 41, BATON-ECB96: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	BATON	Multiple of 12	Same as input length	No final part
C_Decrypt	BATON	Multiple of 12	Same as input length	No final part

## 2.9.6 BATON-CBC128

BATON-CBC128, denoted **CKM\_BATON\_CBC128**, is a mechanism for single- and multiple-part encryption and decryption with BATON in 128-bit cipher-block chaining mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

Table 42, BATON-CBC128

Function	Key type	Input length	Output length	Comments
C_Encrypt	BATON	Multiple of 16	Same as input length	No final part
C_Decrypt	BATON	Multiple of 16	Same as input length	No final part

## 2.9.7 BATON-COUNTER

BATON-COUNTER, denoted **CKM\_BATON\_COUNTER**, is a mechanism for single- and multiple-part encryption and decryption with BATON in counter mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

Table 43, BATON-COUNTER: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	BATON	Multiple of 16	Same as input length	No final part
C_Decrypt	BATON	Multiple of 16	Same as input length	No final part

## 2.9.8 BATON-SHUFFLE

BATON-SHUFFLE, denoted **CKM\_BATON\_SHUFFLE**, is a mechanism for single- and multiple-part encryption and decryption with BATON in shuffle mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table:

Table 44, BATON-SHUFFLE: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	BATON	Multiple of 16	Same as input length	No final part
C_Decrypt	BATON	Multiple of 16	Same as input length	No final part

## 2.9.9 BATON WRAP

The BATON wrap and unwrap mechanism, denoted **CKM\_BATON\_WRAP**, is a function used to wrap and unwrap a secret key (MEK). It MAY wrap and unwrap SKIPJACK, BATON and JUNIPER keys.

It has no parameters.

When used to unwrap a key, this mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to it.

## 2.10 JUNIPER

### 2.10.1 Definitions

This section defines the key type “CKK\_JUNIPER” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

CKM\_JUNIPER\_KEY\_GEN

CKM\_JUNIPER\_ECB128

CKM\_JUNIPER\_CBC128

CKM\_JUNIPER\_COUNTER

CKM\_JUNIPER\_SHUFFLE



1356 CKM\_JUNIPER\_WRAP

1357 **2.10.2 JUNIPER secret key objects**

1358 JUNIPER secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_JUNIPER**) hold single-  
1359 length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to  
1360 the common attributes defined for this object class:

1361 *Table 45, JUNIPER Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (40 bytes long)

1362 Refer to [PKCS #11-Base] table 15 for footnotes

1363  
1364 JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or  
1365 unwrap a BATON key with incorrect checksum bits MUST return an error.  
1366 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key  
1367 with a specified value. Nonetheless, we provide templates for doing so.

1368 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1369 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1370 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1371 CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";  
1372 CK_BYTE value[40] = {...};  
1373 CK_BBOOL true = CK_TRUE;  
1374 CK_ATTRIBUTE template[] = {  
1375     {CKA_CLASS, &class, sizeof(class)},  
1376     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1377     {CKA_TOKEN, &true, sizeof(true)},  
1378     {CKA_LABEL, label, sizeof(label)-1},  
1379     {CKA_ENCRYPT, &true, sizeof(true)},  
1380     {CKA_VALUE, value, sizeof(value)}  
1381 };
```

1382 The following is a sample template for creating a JUNIPER TEK secret key object:

```
1383 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1384 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1385 CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";  
1386 CK_BYTE value[40] = {...};  
1387 CK_BBOOL true = CK_TRUE;  
1388 CK_ATTRIBUTE template[] = {  
1389     {CKA_CLASS, &class, sizeof(class)},  
1390     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1391     {CKA_TOKEN, &true, sizeof(true)},  
1392     {CKA_LABEL, label, sizeof(label)-1},  
1393     {CKA_ENCRYPT, &true, sizeof(true)},  
1394     {CKA_WRAP, &true, sizeof(true)},  
1395     {CKA_VALUE, value, sizeof(value)}  
1396 };
```

1397 **2.10.3 JUNIPER key generation**

1398 The JUNIPER key generation mechanism, denoted **CKM\_JUNIPER\_KEY\_GEN**, is a key generation  
1399 mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).  
1400 It does not have a parameter.

1401 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1402 key.



## 2.10.4 JUNIPER-ECB128

JUNIPER-ECB128, denoted **CKM\_JUNIPER\_ECB128**, is a mechanism for single- and multiple-part encryption and decryption with JUNIPER in 128-bit electronic codebook mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table. For encryption and decryption, the input and output data (parts) MAY begin at the same location in memory.

Table 46, JUNIPER-ECB128: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	JUNIPER	Multiple of 16	Same as input length	No final part
C_Decrypt	JUNIPER	Multiple of 16	Same as input length	No final part

## 2.10.5 JUNIPER-CBC128

JUNIPER-CBC128, denoted **CKM\_JUNIPER\_CBC128**, is a mechanism for single- and multiple-part encryption and decryption with JUNIPER in 128-bit cipher block chaining mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table. For encryption and decryption, the input and output data (parts) MAY begin at the same location in memory.

Table 47, JUNIPER-CBC128: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	JUNIPER	Multiple of 16	Same as input length	No final part
C_Decrypt	JUNIPER	Multiple of 16	Same as input length	No final part

## 2.10.6 JUNIPER-COUNTER

JUNIPER-COUNTER, denoted **CKM\_JUNIPER\_COUNTER**, is a mechanism for single- and multiple-part encryption and decryption with JUNIPER in counter mode.

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application MAY NOT specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

Constraints on key types and the length of data are summarized in the following table. For encryption and decryption, the input and output data (parts) MAY begin at the same location in memory.

Table 48, JUNIPER-COUNTER: Data and Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	JUNIPER	Multiple of 16	Same as input length	No final part
C_Decrypt	JUNIPER	Multiple of 16	Same as input length	No final part

## 2.10.7 JUNIPER-SHUFFLE

JUNIPER-SHUFFLE, denoted **CKM\_JUNIPER\_SHUFFLE**, is a mechanism for single- and multiple-part encryption and decryption with JUNIPER in shuffle mode.

1433 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1434 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1435 encrypting. It MAY, of course, specify a particular IV when decrypting.

1436 Constraints on key types and the length of data are summarized in the following table. For encryption  
1437 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1438 *Table 49, JUNIPER-SHUFFLE: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	JUNIPER	Multiple of 16	Same as input length	No final part
C_Decrypt	JUNIPER	Multiple of 16	Same as input length	No final part

1439 **2.10.8 JUNIPER WRAP**

1440 The JUNIPER wrap and unwrap mechanism, denoted **CKM\_JUNIPER\_WRAP**, is a function used to wrap  
1441 and unwrap an MEK. It MAY wrap or unwrap SKIPJACK, BATON and JUNIPER keys.

1442 It has no parameters.

1443 When used to unwrap a key, this mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and  
1444 **CKA\_VALUE** attributes to it.

1445 **2.11 MD2**

1446 **2.11.1 Definitions**

1447 Mechanisms:

1448 CKM\_MD2

1449 CKM\_MD2\_HMAC

1450 CKM\_MD2\_HMAC\_GENERAL

1451 CKM\_MD2\_KEY\_DERIVATION

1452 **2.11.2 MD2 digest**

1453 The MD2 mechanism, denoted **CKM\_MD2**, is a mechanism for message digesting, following the MD2  
1454 message-digest algorithm defined in RFC 6149.

1455 It does not have a parameter.

1456 Constraints on the length of data are summarized in the following table:

1457 *Table 50, MD2: Data Length*

Function	Data length	Digest Length
C_Digest	Any	16

1458 **2.11.3 General-length MD2-HMAC**

1459 The general-length MD2-HMAC mechanism, denoted **CKM\_MD2\_HMAC\_GENERAL**, is a mechanism for  
1460 signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it  
1461 uses are generic secret keys.

1462 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1463 output. This length should be in the range 0-16 (the output size of MD2 is 16 bytes). Signatures (MACs)  
1464 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

Table 51, General-length MD2-HMAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	Generic secret	Any	0-16, depending on parameters
C_Verify	Generic secret	Any	0-16, depending on parameters

## 2.11.4 MD2-HMAC

The MD2-HMAC mechanism, denoted **CKM\_MD2\_HMAC**, is a special case of the general-length MD2-HMAC mechanism in Section 2.11.3.

It has no parameter, and produces an output of length 16.

## 2.11.5 MD2 key derivation

MD2 key derivation, denoted **CKM\_MD2\_KEY\_DERIVATION**, is a mechanism which provides the capability of deriving a secret key by digesting the value of another secret key with MD2.

The value of the base key is digested once, and the result is used to make the value of the derived secret key.

- If no length or key type is provided in the template, then the key produced by this mechanism MUST be a generic secret key. Its length MUST be 16 bytes (the output size of MD2)..
- If no key type is provided in the template, but a length is, then the key produced by this mechanism MUST be a generic secret key of the specified length.
- If no length was provided in the template, but a key type is, then that key type must have a well-defined length. If it does, then the key produced by this mechanism MUST be of the type specified in the template. If it doesn't, an error MUST be returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by this mechanism MUST be of the specified type and length.

If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set properly.

If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.

This mechanism has the following rules about key sensitivity and extractability:

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key MUST as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

## 2.12 MD5

### 2.12.1 Definitions

Mechanisms:

CKM\_MD5

CKM\_MD5\_HMAC

1504 CKM\_MD5\_HMAC\_GENERAL  
1505 CKM\_MD5\_KEY\_DERIVATION

1506 **2.12.2 MD5 Digest**

1507 The MD5 mechanism, denoted **CKM\_MD5**, is a mechanism for message digesting, following the MD5  
1508 message-digest algorithm defined in RFC 1321.

1509 It does not have a parameter.

1510 Constraints on the length of input and output data are summarized in the following table. For single-part  
1511 digesting, the data and the digest MAY begin at the same location in memory.

1512 *Table 52, MD5: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1513 **2.12.3 General-length MD5-HMAC**

1514 The general-length MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC\_GENERAL**, is a mechanism for  
1515 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it  
1516 uses are generic secret keys.

1517 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1518 output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)  
1519 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1520 *Table 53, General-length MD5-HMAC: Key and Data Length*

Function	Key type	Data length	Signature length
C_Sign	Generic secret	Any	0-16, depending on parameters
C_Verify	Generic secret	Any	0-16, depending on parameters

1521 **2.12.4 MD5-HMAC**

1522 The MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC**, is a special case of the general-length MD5-  
1523 HMAC mechanism in Section 2.12.3.

1524 It has no parameter, and produces an output of length 16.

1525 **2.12.5 MD5 key derivation**

1526 MD5 key derivation denoted **CKM\_MD5\_KEY\_DERIVATION**, is a mechanism which provides the  
1527 capability of deriving a secret key by digesting the value of another secret key with MD5.

1528 The value of the base key is digested once, and the result is used to make the value of derived secret  
1529 key.

- 1530 • If no length or key type is provided in the template, then the key produced by this mechanism MUST  
1531 be a generic secret key. Its length MUST be 16 bytes (the output size of MD5).
- 1532 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
1533 MUST be a generic secret key of the specified length.
- 1534 • If no length was provided in the template, but a key type is, then that key type must have a well-  
1535 defined length. If it does, then the key produced by this mechanism MUST be of the type specified in  
1536 the template. If it doesn't, an error MUST be returned.
- 1537 • If both a key type and a length are provided in the template, the length must be compatible with that  
1538 key type. The key produced by this mechanism MUST be of the specified type and length.

1539 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set  
 1540 properly.

1541 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.

1542 This mechanism has the following rules about key sensitivity and extractability.

- 1543 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY  
 1544 both be specified to either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
 1545 default value.
- 1546 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
 1547 MUST as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then  
 1548 the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
 1549 **CKA\_SENSITIVE** attribute.
- 1550 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
 1551 derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
 1552 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
 1553 value from its **CKA\_EXTRACTABLE** attribute.

1554 **2.13 FASTHASH**

1555 **2.13.1 Definitions**

1556 Mechanisms:  
 1557 CKM\_FASTHASH

1558 **2.13.2 FASTHASH digest**

1559 The FASTHASH mechanism, denoted **CKM\_FASTHASH**, is a mechanism for message digesting,  
 1560 following the U.S. government's algorithm.

1561 It does not have a parameter.

1562 Constraints on the length of input and output data are summarized in the following table:

1563 *Table 54, FASTHASH: Data Length*

Function	Input length	Digest length
C_Digest	Any	40

1564 **2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)**

1565 **2.14.1 Definitions**

1566 The mechanisms in this section are for generating keys and IVs for performing password-based  
 1567 encryption. The method used to generate keys and IVs is specified in PKCS #5.

1568 Mechanisms:

- 1569 CKM\_PBE\_MD2\_DES\_CBC
- 1570 CKM\_PBE\_MD5\_DES\_CBC
- 1571 CKM\_PBE\_MD5\_CAST\_CBC
- 1572 CKM\_PBE\_MD5\_CAST3\_CBC
- 1573 CKM\_PBE\_MD5\_CAST5\_CBC
- 1574 CKM\_PBE\_MD5\_CAST128\_CBC
- 1575 CKM\_PBE\_SHA1\_CAST5\_CBC
- 1576 CKM\_PBE\_SHA1\_CAST128\_CBC

1577 CKM\_PBE\_SHA1\_RC4\_128  
1578 CKM\_PBE\_SHA1\_RC4\_40  
1579 CKM\_PBE\_SHA1\_RC2\_128\_CBC  
1580 CKM\_PBE\_SHA1\_RC2\_40\_CBC

## 1581 2.14.2 Password-based encryption/authentication mechanism parameters

### 1582 2.14.2.1 CK\_PBE\_PARAMS; CK\_PBE\_PARAMS\_PTR

1583 **CK\_PBE\_PARAMS** is a structure which provides all of the necessary information required by the  
1584 CKM\_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation  
1585 mechanisms) and the CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC mechanism. It is defined as follows:

```
1586 typedef struct CK_PBE_PARAMS {  
1587     CK_BYTE_PTR pInitVector;  
1588     CK_UTF8CHAR_PTR pPassword;  
1589     CK_ULONG ulPasswordLen;  
1590     CK_BYTE_PTR pSalt;  
1591     CK_ULONG ulSaltLen;  
1592     CK_ULONG ulIteration;  
1593 } CK_PBE_PARAMS;
```

1594 The fields of the structure have the following meanings:

1595	<i>pInitVector</i>	pointer to the location that receives the 8-byte initialization vector (IV), if an IV is required
1596		
1597	<i>pPassword</i>	points to the password to be used in the PBE key generation
1598		
1599	<i>ulPasswordLen</i>	length in bytes of the password information
1600	<i>pSalt</i>	points to the salt to be used in the PBE key generation
1601	<i>ulSaltLen</i>	length in bytes of the salt information
1602	<i>ulIteration</i>	number of iterations required for the generation

1603 **CK\_PBE\_PARAMS\_PTR** is a pointer to a **CK\_PBE\_PARAMS**.

### 1604 2.14.3 MD2-PBE for DES-CBC

1605 MD2-PBE for DES-CBC, denoted **CKM\_PBE\_MD2\_DES\_CBC**, is a mechanism used for generating a  
1606 DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an  
1607 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1608 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1609 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1610 generated by the mechanism.

### 1611 2.14.4 MD5-PBE for DES-CBC

1612 MD5-PBE for DES-CBC, denoted **CKM\_PBE\_MD5\_DES\_CBC**, is a mechanism used for generating a  
1613 DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an  
1614 iteration count. This functionality is defined in PKCS #5 as PBKDF1.



1615 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1616 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1617 generated by the mechanism.

#### 1618 **2.14.5 MD5-PBE for CAST-CBC**

1619 MD5-PBE for CAST-CBC, denoted **CKM\_PBE\_MD5\_CAST\_CBC**, is a mechanism used for generating a  
1620 CAST secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an  
1621 iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1622 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1623 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1624 generated by the mechanism

1625 The length of the CAST key generated by this mechanism MAY be specified in the supplied template; if it  
1626 is not present in the template, it defaults to 8 bytes.

#### 1627 **2.14.6 MD5-PBE for CAST3-CBC**

1628 MD5-PBE for CAST3-CBC, denoted **CKM\_PBE\_MD5\_CAST3\_CBC**, is a mechanism used for generating  
1629 a CAST3 secret key and an IV from a password and a salt value by using the MD5 digest algorithm and  
1630 an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1631 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1632 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1633 generated by the mechanism

1634 The length of the CAST3 key generated by this mechanism MAY be specified in the supplied template; if  
1635 it is not present in the template, it defaults to 8 bytes.

#### 1636 **2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC)**

1637 MD5-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM\_PBE\_MD5\_CAST128\_CBC** or  
1638 **CKM\_PBE\_MD5\_CAST5\_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key  
1639 and an IV from a password and a salt value by using the MD5 digest algorithm and an iteration count.  
1640 This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1641 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1642 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1643 generated by the mechanism

1644 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied  
1645 template; if it is not present in the template, it defaults to 8 bytes.

#### 1646 **2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)**

1647 SHA-1-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM\_PBE\_SHA1\_CAST128\_CBC** or  
1648 **CKM\_PBE\_SHA1\_CAST5\_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key  
1649 and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This  
1650 functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1651 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1652 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1653 generated by the mechanism

1654 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied  
1655 template; if it is not present in the template, it defaults to 8 bytes



## 2.15 PKCS #12 password-based encryption/authentication mechanisms

### 2.15.1 Definitions

The mechanisms in this section are for generating keys and IVs for performing password-based encryption or authentication. The method used to generate keys and IVs is based on a method that was specified in PKCS #12.

We specify here a general method for producing various types of pseudo-random bits from a password,  $p$ ; a string of salt bits,  $s$ ; and an iteration count,  $c$ . The “type” of pseudo-random bits to be produced is identified by an identification byte,  $ID$ , described at the end of this section.

Let  $H$  be a hash function built around a compression function  $f: \mathbf{Z}_2^u \times \mathbf{Z}_2^v \rightarrow \mathbf{Z}_2^u$  (that is,  $H$  has a chaining variable and output of length  $u$  bits, and the message input to the compression function of  $H$  is  $v$  bits). For MD2 and MD5,  $u=128$  and  $v=512$ ; for SHA-1,  $u=160$  and  $v=512$ .

We assume here that  $u$  and  $v$  are both multiples of 8, as are the lengths in bits of the password and salt strings and the number  $n$  of pseudo-random bits required. In addition,  $u$  and  $v$  are of course nonzero.

1. Construct a string,  $D$  (the “diversifier”), by concatenating  $v/8$  copies of  $ID$ .
2. Concatenate copies of the salt together to create a string  $S$  of length  $v \cdot \lceil s/v \rceil$  bits (the final copy of the salt MAY be truncated to create  $S$ ). Note that if the salt is the empty string, then so is  $S$ .
3. Concatenate copies of the password together to create a string  $P$  of length  $v \cdot \lceil p/v \rceil$  bits (the final copy of the password MAY be truncated to create  $P$ ). Note that if the password is the empty string, then so is  $P$ .
4. Set  $I = S || P$  to be the concatenation of  $S$  and  $P$ .
5. Set  $j = \lceil n/u \rceil$ .
6. For  $i=1, 2, \dots, j$ , do the following:
  - a. Set  $A_i = H_c(D || I)$ , the  $i$ th hash of  $D || I$ . That is, compute the hash of  $D || I$ ; compute the hash of that hash; etc.; continue in this fashion until a total of  $c$  hashes have been computed, each on the result of the previous hash.
  - b. Concatenate copies of  $A_i$  to create a string  $B$  of length  $v$  bits (the final copy of  $A_i$  MAY be truncated to create  $B$ ).
  - c. Treating  $I$  as a concatenation  $I_0, I_1, \dots, I_{k-1}$  of  $v$ -bit blocks, where  $k = \lceil s/v \rceil + \lceil p/v \rceil$ , modify  $I$  by setting  $I_j = (I_j + B + 1) \bmod 2^v$  for each  $j$ . To perform this addition, treat each  $v$ -bit block as a binary number represented most-significant bit first.
7. Concatenate  $A_1, A_2, \dots, A_j$  together to form a pseudo-random bit string,  $A$ .
8. Use the first  $n$  bits of  $A$  as the output of this entire process.

When the password-based encryption mechanisms presented in this section are used to generate a key and IV (if needed) from a password, salt, and an iteration count, the above algorithm is used. To generate a key, the identifier byte  $ID$  is set to the value 1; to generate an IV, the identifier byte  $ID$  is set to the value 2.

When the password-based authentication mechanism presented in this section is used to generate a key from a password, salt and an iteration count, the above algorithm is used. The identifier  $ID$  is set to the value 3.

### 2.15.2 SHA-1-PBE for 128-bit RC4

SHA-1-PBE for 128-bit RC4, denoted **CKM\_PBE\_SHA1\_RC4\_128**, is a mechanism used for generating a 128-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an iteration count. The method used to generate the key is described above.

It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the key generation process. The parameter also has a field to hold the location of an application-supplied

buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not require an IV.

The key produced by this mechanism will typically be used for performing password-based encryption.

### 2.15.3 SHA-1\_PBE for 40-bit RC4

SHA-1-PBE for 40-bit RC4, denoted **CKM\_PBE\_SHA1\_RC4\_40**, is a mechanism used for generating a 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an iteration count. The method used to generate the key is described above.

It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the key generation process. The parameter also has a field to hold the location of an application-supplied buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not require an IV.

The key produced by this mechanism will typically be used for performing password-based encryption.

### 2.15.4 SHA-1\_PBE for 128-bit RC2-CBC

SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_128\_CBC**, is a mechanism used for generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm and an iteration count. The method used to generate the key and IV is described above.

It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the key generation process and the location of an application-supplied buffer which receives the 8-byte IV generated by the mechanism.

When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object Identifier `pbeWithSHA1And128BitRC2-CBC`.

The key and IV produced by this mechanism will typically be used for performing password-based encryption.

### 2.15.5 SHA-1\_PBE for 40-bit RC2-CBC

SHA-1-PBE for 40-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_40\_CBC**, is a mechanism used for generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm and an iteration count. The method used to generate the key and IV is described above.

It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the key generation process and the location of an application-supplied buffer which receives the 8-byte IV generated by the mechanism.

When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object Identifier `pbeWithSHA1And40BitRC2-CBC`.

The key and IV produced by this mechanism will typically be used for performing password-based encryption

## 2.16 RIPE-MD

### 2.16.1 Definitions

Mechanisms:

**CKM\_RIPEMD128**

**CKM\_RIPEMD128\_HMAC**

**CKM\_RIPEMD128\_HMAC\_GENERAL**

**CKM\_RIPEMD160**

1745 CKM\_RIPEMD160\_HMAC  
1746 CKM\_RIPEMD160\_HMAC\_GENERAL

## 1747 2.16.2 RIPE-MD 128 Digest

1748 The RIPE-MD 128 mechanism, denoted **CKM\_RIPEMD128**, is a mechanism for message digesting,  
1749 following the RIPE-MD 128 message-digest algorithm.

1750 It does not have a parameter.

1751 Constraints on the length of data are summarized in the following table:

1752 *Table 55, RIPE-MD 128: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1753

## 1754 2.16.3 General-length RIPE-MD 128-HMAC

1755 The general-length RIPE-MD 128-HMAC mechanism, denoted **CKM\_RIPEMD128\_HMAC\_GENERAL**, is  
1756 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 128  
1757 hash function. The keys it uses are generic secret keys.

1758 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1759 output. This length should be in the range 0-16 (the output size of RIPE-MD 128 is 16 bytes). Signatures  
1760 (MACs) produced by this mechanism **MUST** be taken from the start of the full 16-byte HMAC output.

1761 *Table 56, General-length RIPE-MD 128-HMAC*

Function	Key type	Data length	Signature length
C_Sign	Generic secret	Any	0-16, depending on parameters
C_Verify	Generic secret	Any	0-16, depending on parameters

## 1762 2.16.4 RIPE-MD 128-HMAC

1763 The RIPE-MD 128-HMAC mechanism, denoted **CKM\_RIPEMD128\_HMAC**, is a special case of the  
1764 general-length RIPE-MD 128-HMAC mechanism in Section 2.16.3.

1765 It has no parameter, and produces an output of length 16.

## 1766 2.16.5 RIPE-MD 160

1767 The RIPE-MD 160 mechanism, denoted **CKM\_RIPEMD160**, is a mechanism for message digesting,  
1768 following the RIPE-MD 160 message-digest defined in ISO-10118.

1769 It does not have a parameter.

1770 Constraints on the length of data are summarized in the following table:

1771 *Table 57, RIPE-MD 160: Data Length*

Function	Data length	Digest length
C_Digest	Any	20

## 2.16.6 General-length RIPE-MD 160-HMAC

The general-length RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC\_GENERAL**, is a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160 hash function. The keys it uses are generic secret keys.

It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures (MACs) produced by this mechanism MUST be taken from the start of the full 20-byte HMAC output.

*Table 58, General-length RIPE-MD 160-HMAC: Data and Length*

Function	Key type	Data length	Signature length
C_Sign	Generic secret	Any	0-20, depending on parameters
C_Verify	Generic secret	Any	0-20, depending on parameters

## 2.16.7 RIPE-MD 160-HMAC

The RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC**, is a special case of the general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.

It has no parameter, and produces an output of length 20.

## 2.17 SET

### 2.17.1 Definitions

Mechanisms:

**CKM\_KEY\_WRAP\_SET\_OAEP**

### 2.17.2 SET mechanism parameters

#### 2.17.2.1 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS; CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR

**CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** is a structure that provides the parameters to the **CKM\_KEY\_WRAP\_SET\_OAEP** mechanism. It is defined as follows:

```
typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {  
    CK_BYTE bBC;  
    CK_BYTE_PTR pX;  
    CK_ULONG ulXLen;  
} CK_KEY_WRAP_SET_OAEP_PARAMS;
```

The fields of the structure have the following meanings:

*bBC*      block contents byte

*pX*      concatenation of hash of plaintext data (if present) and  
extra data (if present)

*ulXLen*      length in bytes of concatenation of hash of plaintext data  
(if present) and extra data (if present). 0 if neither is  
present.

**CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR** is a pointer to a  
**CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS**.

### 2.17.3 OAEP key wrapping for SET

The OAEP key wrapping for SET mechanism, denoted **CKM\_KEY\_WRAP\_SET\_OAEP**, is a mechanism for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some extra data MAY be wrapped together with the DES key. This mechanism is defined in the SET protocol specifications.

It takes a parameter, a **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure. This structure holds the "Block Contents" byte of the data and the concatenation of the hash of plaintext data (if present) and the extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by the *ulXLen* field having the value 0.

When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present) and the extra data (if present) is returned following the convention described [PKCS #11-Curr], **Miscellaneous simple key derivation mechanisms**. Note that if the inputs to **C\_UnwrapKey** are such that the extra data is not returned (e.g. the buffer supplied in the **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure is **NULL\_PTR**), then the unwrapped key object MUST NOT be created, either.

Be aware that when this mechanism is used to unwrap a key, the *bBC* and *pX* fields of the parameter supplied to the mechanism MAY be modified.

If an application uses **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP**, it may be preferable for it simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data (this concatenation MUST NOT be larger than 128 bytes), rather than calling **C\_UnwrapKey** twice. Each call of **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP** requires an RSA decryption operation to be performed, and this computational overhead MAY be avoided by this means.

## 2.18 LYNKS

### 2.18.1 Definitions

Mechanisms:

**CKM\_KEY\_WRAP\_LYNKS**

### 2.18.2 LYNKS key wrapping

The LYNKS key wrapping mechanism, denoted **CKM\_KEY\_WRAP\_LYNKS**, is a mechanism for wrapping and unwrapping secret keys with DES keys. It MAY wrap any 8-byte secret key, and it produces a 10-byte wrapped key, containing a cryptographic checksum.

It does not have a parameter.

To wrap an 8-byte secret key *K* with a DES key *W*, this mechanism performs the following steps:

1. Initialize two 16-bit integers,  $sum_1$  and  $sum_2$ , to 0
2. Loop through the bytes of *K* from first to last.
3. Set  $sum_1 = sum_1 + \text{the key byte}$  (treat the key byte as a number in the range 0-255).
4. Set  $sum_2 = sum_2 + sum_1$ .
5. Encrypt *K* with *W* in ECB mode, obtaining an encrypted key, *E*.
6. Concatenate the last 6 bytes of *E* with  $sum_2$ , representing  $sum_2$  most-significant bit first. The result is an 8-byte block, *T*
7. Encrypt *T* with *W* in ECB mode, obtaining an encrypted checksum, *C*.
8. Concatenate *E* with the last 2 bytes of *C* to obtain the wrapped key.

When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly, an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.

---

### 3 PKCS #11 Implementation Conformance

1852

1853 An implementation is a conforming implementation if it meets the conditions specified in one or more  
1854 server profiles specified in **[PKCS #11-Prof]**.

1855 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1856 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL  
1857 conform to all normative statements within the clauses specified for that profile and for any subclauses to  
1858 each of those clauses .

1859

---

## Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

### Participants:

Gil Abel, Athena Smartcard Solutions, Inc.  
Warren Armstrong, QuintessenceLabs  
Jeff Bartell, Semper Foris Solutions LLC  
Peter Bartok, Venafi, Inc.  
Anthony Berglas, Cryptsoft  
Joseph Brand, Semper Fortis Solutions LLC  
Kelley Burgin, National Security Agency  
Robert Burns, Thales e-Security  
Wan-Teh Chang, Google Inc.  
Hai-May Chao, Oracle  
Janice Cheng, Vormetric, Inc.  
Sangrae Cho, Electronics and Telecommunications Research Institute (ETRI)  
Doron Cohen, SafeNet, Inc.  
Fadi Cotran, Futurex  
Tony Cox, Cryptsoft  
Christopher Duane, EMC  
Chris Dunn, SafeNet, Inc.  
Valerie Fenwick, Oracle  
Terry Fletcher, SafeNet, Inc.  
Susan Gleeson, Oracle  
Sven Gossel, Charismathics  
John Green, QuintessenceLabs  
Robert Griffin, EMC  
Paul Grojean, Individual  
Peter Gutmann, Individual  
Dennis E. Hamilton, Individual  
Thomas Hardjono, M.I.T.  
Tim Hudson, Cryptsoft  
Gershon Janssen, Individual  
Seunghun Jin, Electronics and Telecommunications Research Institute (ETRI)  
Wang Jingman, Feitan Technologies  
Andrey Jivsov, Symantec Corp.  
Mark Joseph, P6R  
Stefan Kaesar, Infineon Technologies  
Greg Kazmierczak, Wave Systems Corp.



1900 Mark Knight, Thales e-Security  
1901 Darren Krahn, Google Inc.  
1902 Alex Krasnov, Infineon Technologies AG  
1903 Dina Kurktchi-Nimeh, Oracle  
1904 Mark Lambiase, SecureAuth Corporation  
1905 Lawrence Lee, GoTrust Technology Inc.  
1906 John Leiseboer, QuintessenceLabs  
1907 Sean Leon, Infineon Technologies  
1908 Geoffrey Li, Infineon Technologies  
1909 Howie Liu, Infineon Technologies  
1910 Hal Lockhart, Oracle  
1911 Robert Lockhart, Thales e-Security  
1912 Dale Moberg, Axway Software  
1913 Darren Moffat, Oracle  
1914 Valery Osheter, SafeNet, Inc.  
1915 Sean Parkinson, EMC  
1916 Rob Philpott, EMC  
1917 Mark Powers, Oracle  
1918 Ajai Puri, SafeNet, Inc.  
1919 Robert Relyea, Red Hat  
1920 Saikat Saha, Oracle  
1921 Subhash Sankuratipati, NetApp  
1922 Anthony Scarpino, Oracle  
1923 Johann Schoetz, Infineon Technologies AG  
1924 Rayees Shamsuddin, Wave Systems Corp.  
1925 Radhika Siravara, Oracle  
1926 Brian Smith, Mozilla Corporation  
1927 David Smith, Venafi, Inc.  
1928 Ryan Smith, Futurex  
1929 Jerry Smith, US Department of Defense (DoD)  
1930 Oscar So, Oracle  
1931 Graham Steel, Cryptosense  
1932 Michael Stevens, QuintessenceLabs  
1933 Michael StJohns, Individual  
1934 Jim Susoy, P6R  
1935 Sander Temme, Thales e-Security  
1936 Kiran Thota, VMware, Inc.  
1937 Walter-John Turnes, Gemini Security Solutions, Inc.  
1938 Stef Walter, Red Hat  
1939 James Wang, Vormetric  
1940 Jeff Webb, Dell  
1941 Peng Yu, Feitian Technologies

- 1942 Magda Zdunkiewicz, Cryptsoft
- 1943 Chris Zimman, Individual

---

## Appendix B. Manifest constants

The following constants have been defined for PKCS #11 V2.40. Also, refer to **[PKCS #11-Base]** and **[PKCS #11-Curr]** for additional definitions.

```
/*
 * Copyright OASIS Open 2014. All rights reserved.
 * OASIS trademark, IPR and other policies apply.
 * http://www.oasis-open.org/policies-guidelines/ipr
 */

#define CKK_KEA 0x00000005
#define CKK_RC2 0x00000011
#define CKK_RC4 0x00000012
#define CKK_DES 0x00000013
#define CKK_CAST 0x00000016
#define CKK_CAST3 0x00000017
#define CKK_CAST5 0x00000018
#define CKK_CAST128 0x00000018
#define CKK_RC5 0x00000019
#define CKK_IDEA 0x0000001A
#define CKK_SKIPJACK 0x0000001B
#define CKK_BATON 0x0000001C
#define CKK_JUNIPER 0x0000001D
#define CKM_MD2_RSA_PKCS 0x00000004
#define CKM_MD5_RSA_PKCS 0x00000005
#define CKM_RIPEMD128_RSA_PKCS 0x00000007
#define CKM_RIPEMD160_RSA_PKCS 0x00000008
#define CKM_RC2_KEY_GEN 0x00000100
#define CKM_RC2_ECB 0x00000101
#define CKM_RC2_CBC 0x00000102
#define CKM_RC2_MAC 0x00000103
#define CKM_RC2_MAC_GENERAL 0x00000104
#define CKM_RC2_CBC_PAD 0x00000105
#define CKM_RC4_KEY_GEN 0x00000110
#define CKM_RC4 0x00000111
#define CKM_DES_KEY_GEN 0x00000120
#define CKM_DES_ECB 0x00000121
#define CKM_DES_CBC 0x00000122
#define CKM_DES_MAC 0x00000123
#define CKM_DES_MAC_GENERAL 0x00000124
#define CKM_DES_CBC_PAD 0x00000125
#define CKM_MD2 0x00000200
#define CKM_MD2_HMAC 0x00000201
#define CKM_MD2_HMAC_GENERAL 0x00000202
#define CKM_MD5 0x00000210
#define CKM_MD5_HMAC 0x00000211
#define CKM_MD5_HMAC_GENERAL 0x00000212
#define CKM_RIPEMD128 0x00000230
#define CKM_RIPEMD128_HMAC 0x00000231
#define CKM_RIPEMD128_HMAC_GENERAL 0x00000232
#define CKM_RIPEMD160 0x00000240
#define CKM_RIPEMD160_HMAC 0x00000241
#define CKM_RIPEMD160_HMAC_GENERAL 0x00000242
#define CKM_CAST_KEY_GEN 0x00000300
#define CKM_CAST_ECB 0x00000301
#define CKM_CAST_CBC 0x00000302
#define CKM_CAST_MAC 0x00000303
#define CKM_CAST_MAC_GENERAL 0x00000304
#define CKM_CAST_CBC_PAD 0x00000305
#define CKM_CAST3_KEY_GEN 0x00000310
```

```

2003 #define CKM_CAST3_ECB 0x00000311
2004 #define CKM_CAST3_CBC 0x00000312
2005 #define CKM_CAST3_MAC 0x00000313
2006 #define CKM_CAST3_MAC_GENERAL 0x00000314
2007 #define CKM_CAST3_CBC_PAD 0x00000315
2008 #define CKM_CAST5_KEY_GEN 0x00000320
2009 #define CKM_CAST128_KEY_GEN 0x00000320
2010 #define CKM_CAST5_ECB 0x00000321
2011 #define CKM_CAST128_ECB 0x00000321
2012 #define CKM_CAST5_CBC 0x00000322
2013 #define CKM_CAST128_CBC 0x00000322
2014 #define CKM_CAST5_MAC 0x00000323
2015 #define CKM_CAST128_MAC 0x00000323
2016 #define CKM_CAST5_MAC_GENERAL 0x00000324
2017 #define CKM_CAST128_MAC_GENERAL 0x00000324
2018 #define CKM_CAST5_CBC_PAD 0x00000325
2019 #define CKM_CAST128_CBC_PAD 0x00000325
2020 #define CKM_RC5_KEY_GEN 0x00000330
2021 #define CKM_RC5_ECB 0x00000331
2022 #define CKM_RC5_CBC 0x00000332
2023 #define CKM_RC5_MAC 0x00000333
2024 #define CKM_RC5_MAC_GENERAL 0x00000334
2025 #define CKM_RC5_CBC_PAD 0x00000335
2026 #define CKM_IDEA_KEY_GEN 0x00000340
2027 #define CKM_IDEA_ECB 0x00000341
2028 #define CKM_IDEA_CBC 0x00000342
2029 #define CKM_IDEA_MAC 0x00000343
2030 #define CKM_IDEA_MAC_GENERAL 0x00000344
2031 #define CKM_IDEA_CBC_PAD 0x00000345
2032 #define CKM_MD5_KEY_DERIVATION 0x00000390
2033 #define CKM_MD2_KEY_DERIVATION 0x00000391
2034 #define CKM_PBE_MD2_DES_CBC 0x000003A0
2035 #define CKM_PBE_MD5_DES_CBC 0x000003A1
2036 #define CKM_PBE_MD5_CAST_CBC 0x000003A2
2037 #define CKM_PBE_MD5_CAST3_CBC 0x000003A3
2038 #define CKM_PBE_MD5_CAST5_CBC 0x000003A4
2039 #define CKM_PBE_MD5_CAST128_CBC 0x000003A4
2040 #define CKM_PBE_SHA1_CAST5_CBC 0x000003A5
2041 #define CKM_PBE_SHA1_CAST128_CBC 0x000003A5
2042 #define CKM_PBE_SHA1_RC4_128 0x000003A6
2043 #define CKM_PBE_SHA1_RC4_40 0x000003A7
2044 #define CKM_PBE_SHA1_RC2_128_CBC 0x000003AA
2045 #define CKM_PBE_SHA1_RC2_40_CBC 0x000003AB
2046 #define CKM_KEY_WRAP_LYNKS 0x00000400
2047 #define CKM_KEY_WRAP_SET_OAEP 0x00000401
2048 #define CKM_SKIPJACK_KEY_GEN 0x00001000
2049 #define CKM_SKIPJACK_ECB64 0x00001001
2050 #define CKM_SKIPJACK_CBC64 0x00001002
2051 #define CKM_SKIPJACK_OFB64 0x00001003
2052 #define CKM_SKIPJACK_CFB64 0x00001004
2053 #define CKM_SKIPJACK_CFB32 0x00001005
2054 #define CKM_SKIPJACK_CFB16 0x00001006
2055 #define CKM_SKIPJACK_CFB8 0x00001007
2056 #define CKM_SKIPJACK_WRAP 0x00001008
2057 #define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009
2058 #define CKM_SKIPJACK_RELAYX 0x0000100a
2059 #define CKM_KEA_KEY_PAIR_GEN 0x00001010
2060 #define CKM_KEA_KEY_DERIVE 0x00001011
2061 #define CKM_FORTEZZA_TIMESTAMP 0x00001020
2062 #define CKM_BATON_KEY_GEN 0x00001030
2063 #define CKM_BATON_ECB128 0x00001031
2064 #define CKM_BATON_ECB96 0x00001032
2065 #define CKM_BATON_CBC128 0x00001033
2066 #define CKM_BATON_COUNTER 0x00001034

```

2067 #define CKM\_BATON\_SHUFFLE 0x00001035  
2068 #define CKM\_BATON\_WRAP 0x00001036  
2069 #define CKM\_JUNIPER\_KEY\_GEN 0x00001060  
2070 #define CKM\_JUNIPER\_ECB128 0x00001061  
2071 #define CKM\_JUNIPER\_CBC128 0x00001062  
2072 #define CKM\_JUNIPER\_COUNTER 0x00001063  
2073 #define CKM\_JUNIPER\_SHUFFLE 0x00001064  
2074 #define CKM\_JUNIPER\_WRAP 0x00001065  
2075 #define CKM\_FASTHASH 0x00001070

2076

---

## Appendix C. Revision History

Revision	Date	Editor	Changes Made
wd01	May 16, 2013	Susan Gleeson	Initial Template import
wd02	July 7, 2013	Susan Gleeson	Fix references, add participants list, minor cleanup
wd03	October 27, 2013	Robert Griffin	Final participant list and other editorial changes for Committee Specification Draft
csd01	October 30, 2013	OASIS	Committee Specification Draft
wd04	February 19, 2014	Susan Gleeson	Incorporate changes from v2.40 public review
wd05	February 20, 2014	Susan Gleeson	Regenerate table of contents (oversight from wd04)
WD06	February 21, 2014	Susan Gleeson	Remove CKM_PKCS5_PBKD2 from the mechanisms in Table 1.
csd02	April 23, 2014	OASIS	Committee Specification Draft
csd02a	Sep 3 2013	Robert Griffin	Updated revision history and participant list in preparation for Committee Specification ballot