



PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40 **Plus Errata 01**

OASIS Standard **Incorporating Approved Errata 01**

13 May 2016

Specification URIs

This version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/os/pkcs11-hist-v2.40-errata01-os-complete.doc> (Authoritative)
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/os/pkcs11-hist-v2.40-errata01-os-complete.html>
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/os/pkcs11-hist-v2.40-errata01-os-complete.pdf>

Previous version:

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

Valerie Fenwick (valerie.fenwick@oracle.com), Oracle
Robert Relyea (rrelyea@redhat.com), Red Hat

Editors:

Susan Gleeson (susan.gleeson@oracle.com), Oracle
Chris Zimman (chris@wmpp.com), Individual
Robert Griffin (robert.griffin@emc.com), EMC Corporation
Tim Hudson (tjh@cryptsoft.com), Cryptsoft Pty Ltd

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- *PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40 Errata 01*. Edited by Robert Griffin and Tim Hudson. 13 May 2016. OASIS Approved Errata. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/os/pkcs11-hist-v2.40-errata01-os.html>.

Related work:

This specification replaces or supersedes:

- *PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40*. Edited by Susan Gleeson and Chris Zimman. 14 April 2015. OASIS Standard. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/os/pkcs11-hist-v2.40-os.html>.

This specification is related to:

- Normative computer language definition files for PKCS #11 v2.40:
 - <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/os/include/pkcs11-v2.40/pkcs11.h>
 - <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/os/include/pkcs11-v2.40/pkcs11t.h>
 - <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/os/include/pkcs11-v2.40/pkcs11f.h>
- *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>.
- *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40 Plus Errata 01*. Edited by Susan Gleeson, Chris Zimman, Robert Griffin, and Tim Hudson. <http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/errata01/os/pkcs11-curr-v2.40-errata01-os-complete.html>.
- *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40 Errata 01*. Edited by Robert Griffin and Tim Hudson. <http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/errata01/os/pkcs11-curr-v2.40-errata01-os.html>.
- *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40 Plus Errata 01*. Edited by Susan Gleeson, Chris Zimman, Robert Griffin, and Tim Hudson. <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/os/pkcs11-base-v2.40-errata01-os-complete.html>.
- *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40 Errata01*. Edited by Robert Griffin and Tim Hudson. <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/os/pkcs11-base-v2.40-errata01-os.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>.

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 membership of OASIS 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.

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 Plus Errata 01. Edited by Susan Gleeson, Chris Zimman, Robert Griffin and Tim Hudson. 13 May 2016. OASIS Standard Incorporating Approved Errata 01. <http://docs.oasis->

open.org/pkcs11/pkcs11-hist/v2.40/errata01/os/pkcs11-hist-v2.40-errata01-os-complete.html.
Latest version: <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>.

Notices

Copyright © OASIS Open 2016. 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.....	9
1.1	Description of this Document.....	9
1.2	Terminology.....	9
1.3	Definitions.....	9
1.4	Normative References.....	10
1.5	Non-Normative References.....	10
2	Mechanisms.....	13
2.1	PKCS #11 Mechanisms.....	13
2.2	FORTEZZA timestamp.....	16
2.3	KEA.....	16
2.3.1	Definitions.....	16
2.3.2	KEA mechanism parameters.....	16
2.3.2.1	CK_KEA_DERIVE_PARAMS; CK_KEA_DERIVE_PARAMS_PTR.....	16
2.3.3	KEA public key objects.....	17
2.3.4	KEA private key objects.....	18
2.3.5	KEA key pair generation.....	18
2.3.6	KEA key derivation.....	19
2.4	RC2.....	20
2.4.1	Definitions.....	20
2.4.2	RC2 secret key objects.....	20
2.4.3	RC2 mechanism parameters.....	21
2.4.3.1	CK_RC2_PARAMS; CK_RC2_PARAMS_PTR.....	21
2.4.3.2	CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR.....	21
2.4.3.3	CK_RC2_MAC_GENERAL_PARAMS; CK_RC2_MAC_GENERAL_PARAMS_PTR.....	21
2.4.4	RC2 key generation.....	22
2.4.5	RC2-ECB.....	22
2.4.6	RC2-CBC.....	23
2.4.7	RC2-CBC with PKCS padding.....	23
2.4.8	General-length RC2-MAC.....	24
2.4.9	RC2-MAC.....	24
2.5	RC4.....	25
2.5.1	Definitions.....	25
2.5.2	RC4 secret key objects.....	25
2.5.3	RC4 key generation.....	25
2.5.4	RC4 mechanism.....	26
2.6	RC5.....	26
2.6.1	Definitions.....	26
2.6.2	RC5 secret key objects.....	26
2.6.3	RC5 mechanism parameters.....	27
2.6.3.1	CK_RC5_PARAMS; CK_RC5_PARAMS_PTR.....	27
2.6.3.2	CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR.....	27
2.6.3.3	CK_RC5_MAC_GENERAL_PARAMS; CK_RC5_MAC_GENERAL_PARAMS_PTR.....	27
2.6.4	RC5 key generation.....	28
2.6.5	RC5-ECB.....	28

2.6.6 RC5-CBC.....	29
2.6.7 RC5-CBC with PKCS padding	29
2.6.8 General-length RC5-MAC	30
2.6.9 RC5-MAC	30
2.7 General block cipher.....	31
2.7.1 Definitions.....	31
2.7.2 DES secret key objects	32
2.7.3 CAST secret key objects	33
2.7.4 CAST3 secret key objects	33
2.7.5 CAST128 (CAST5) secret key objects	34
2.7.6 IDEA secret key objects	34
2.7.7 CDMF secret key objects	35
2.7.8 General block cipher mechanism parameters.....	35
2.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR.....	35
2.7.9 General block cipher key generation.....	35
2.7.10 General block cipher ECB.....	36
2.7.11 General block cipher CBC.....	36
2.7.12 General block cipher CBC with PCKS padding.....	37
2.7.13 General-length general block cipher MAC	38
2.7.14 General block cipher MAC	38
2.8 SKIPJACK.....	39
2.8.1 Definitions.....	39
2.8.2 SKIPJACK secret key objects	39
2.8.3 SKIPJACK Mechanism parameters	40
2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR	40
2.8.3.2 CK_SKIPJACK_RELAYX_PARAMS; CK_SKIPJACK_RELAYX_PARAMS_PTR	41
2.8.4 SKIPJACK key generation	42
2.8.5 SKIPJACK-ECB64	42
2.8.6 SKIPJACK-CBC64	42
2.8.7 SKIPJACK-OFB64	42
2.8.8 SKIPJACK-CFB64.....	43
2.8.9 SKIPJACK-CFB32.....	43
2.8.10 SKIPJACK-CFB16.....	43
2.8.11 SKIPJACK-CFB8.....	44
2.8.12 SKIPJACK-WRAP	44
2.8.13 SKIPJACK-PRIVATE-WRAP	44
2.8.14 SKIPJACK-RELAYX.....	44
2.9 BATON.....	44
2.9.1 Definitions.....	44
2.9.2 BATON secret key objects	45
2.9.3 BATON key generation	45
2.9.4 BATON-ECB128	46
2.9.5 BATON-ECB96.....	46
2.9.6 BATON-CBC128	46
2.9.7 BATON-COUNTER	47
2.9.8 BATON-SHUFFLE	47

2.9.9 BATON WRAP	47
2.10 JUNIPER.....	47
2.10.1 Definitions.....	47
2.10.2 JUNIPER secret key objects	48
2.10.3 JUNIPER key generation	48
2.10.4 JUNIPER-ECB128	49
2.10.5 JUNIPER-CBC128	49
2.10.6 JUNIPER-COUNTER	49
2.10.7 JUNIPER-SHUFFLE	49
2.10.8 JUNIPER WRAP	50
2.11 MD2	50
2.11.1 Definitions.....	50
2.11.2 MD2 digest	50
2.11.3 General-length MD2-HMAC	50
2.11.4 MD2-HMAC	51
2.11.5 MD2 key derivation.....	51
2.12 MD5	51
2.12.1 Definitions.....	51
2.12.2 MD5 Digest.....	52
2.12.3 General-length MD5-HMAC	52
2.12.4 MD5-HMAC	52
2.12.5 MD5 key derivation.....	52
2.13 FASTHASH.....	53
2.13.1 Definitions.....	53
2.13.2 FASTHASH digest.....	53
2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)	53
2.14.1 Definitions.....	53
2.14.2 Password-based encryption/authentication mechanism parameters.....	54
2.14.2.1 CK_PBE_PARAMS; CK_PBE_PARAMS_PTR	54
2.14.3 MD2-PBE for DES-CBC	54
2.14.4 MD5-PBE for DES-CBC	54
2.14.5 MD5-PBE for CAST-CBC.....	55
2.14.6 MD5-PBE for CAST3-CBC.....	55
2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC).....	55
2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC).....	55
2.15 PKCS #12 password-based encryption/authentication mechanisms	56
2.15.1 Definitions.....	56
2.15.2 SHA-1-PBE for 128-bit RC4	56
2.15.3 SHA-1_PBE for 40-bit RC4	57
2.15.4 SHA-1_PBE for 128-bit RC2-CBC	57
2.15.5 SHA-1_PBE for 40-bit RC2-CBC	57
2.16 RIPE-MD.....	57
2.16.1 Definitions.....	57
2.16.2 RIPE-MD 128 Digest	58
2.16.3 General-length RIPE-MD 128-HMAC	58

2.16.4 RIPE-MD 128-HMAC	58
2.16.5 RIPE-MD 160	58
2.16.6 General-length RIPE-MD 160-HMAC	59
2.16.7 RIPE-MD 160-HMAC	59
2.17 SET	59
2.17.1 Definitions	59
2.17.2 SET mechanism parameters	59
2.17.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS; CK_KEY_WRAP_SET_OAEP_PARAMS_PTR	59
2.17.3 OAEP key wrapping for SET	60
2.18 LYNKS	60
2.18.1 Definitions	60
2.18.2 LYNKS key wrapping	60
3 PKCS #11 Implementation Conformance	61
Appendix A. Acknowledgments	62
Appendix B. Manifest constants	65
Appendix C. Revision History	68

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

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

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

46 1.4 Normative References

47	[PKCS #11-Base]	<i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i>
48		Edited by Susan Gleeson and Chris Zimman. Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html .
49		
50	[PKCS #11-Curr]	<i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40.</i>
51		Edited by Susan Gleeson and Chris Zimman. Latest version.
52		http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html .
53	[PKCS #11-Prof]	<i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i>
54		Edited by Tim Hudson. Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html .
55		
56	[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt .
57		

58 1.5 Non-Normative References

59	[ANSI C]	ANSI/ISO. American National Standard for Programming Languages – C. 1990
60	[ANSI X9.31]	Accredited Standards Committee X9. Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA). 1998.
61		
62	[ANSI X9.42]	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography. 2003
63		
64		
65	[ANSI X9.62]	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998
66		
67	[CC/PP]	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: http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/
68		
69		
70		
71	[CDPD]	Ameritech Mobile Communications et al. Cellular Digital Packet Data System Specifications: Part 406: Airlink Security. 1993
72		
73	[FIPS PUB 46-3]	NIST. <i>FIPS 46-3: Data Encryption Standard (DES).</i> October 26, 2999. URL: http://csrc.nist.gov/publications/fips/index.html
74		
75	[FIPS PUB 81]	NIST. <i>FIPS 81: DES Modes of Operation.</i> December 1980. URL: http://csrc.nist.gov/publications/fips/index.html
76		
77	[FIPS PUB 113]	NIST. <i>FIPS 113: Computer Data Authentication.</i> May 30, 1985. URL: http://csrc.nist.gov/publications/fips/index.html
78		
79	[FIPS PUB 180-2]	NIST. <i>FIPS 180-2: Secure Hash Standard.</i> August 1, 2002. URL: http://csrc.nist.gov/publications/fips/index.html
80		
81	[FORTEZZA CIPG]	NSA, Workstation Security Products. <i>FORTEZZA Cryptologic Interface Programmers Guide, Revision 1.52.</i> November 1985
82		
83	[GCS-API]	X/Open Company Ltd. Generic Cryptographic Service API (GCS-API), Base – Draft 2. February 14, 1995.
84		

- 85 **[ISO/IEC 7816-1]** ISO/IEC 7816-1:2011. *Identification Cards – Integrated circuit cards -- Part 1:*
86 *Cards with contacts -- Physical Characteristics.* 2011 URL:
87 http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089.
- 88 **[ISO/IEC 7816-4]** ISO/IEC 7618-4:2013. *Identification Cards – Integrated circuit cards – Part 4:*
89 *Organization, security and commands for interchange.* 2013. URL:
90 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54550.
91
- 92 **[ISO/IEC 8824-1]** ISO/IEC 8824-1:2008. *Abstract Syntax Notation One (ASN.1): Specification of*
93 *Base Notation.* 2002. URL:
94 http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012
95
- 96 **[ISO/IEC 8825-1]** ISO/IEC 8825-1:2008. Information Technology – ASN.1 Encoding Rules:
97 Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER),
98 and Distinguished Encoding Rules (DER). 2008. URL:
99 http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60
100
- 101 **[ISO/IEC 9594-1]** ISO/IEC 9594-1:2008. *Information Technology – Open System Interconnection –*
102 *The Directory: Overview of Concepts, Models and Services.* 2008. URL:
103 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53364
104
- 105 **[ISO/IEC 9594-8]** ISO/IEC 9594-8:2008. *Information Technology – Open Systems Interconnection*
106 *– The Directory: Public-key and Attribute Certificate Frameworks.* 2008 URL:
107 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53372
108
- 109 **[ISO/IEC 9796-2]** ISO/IEC 9796-2:2010. Information Technology – Security Techniques – Digital
110 Signature Scheme Giving Message Recovery – Part 2: Integer factorization
111 based mechanisms. 2010. URL:
112 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788
113
- 114 **[Java MIDP]** Java Community Process. *Mobile Information Device Profile for Java 2 Micro*
115 *Edition.* November 2002. URL: <http://jcp.org/jsr/detail/118.jsp>
- 116 **[MeT-PTD]** MeT. *MeT PTD Definition – Personal Trusted Device Definition, Version 1.0.*
117 February 2003. URL: <http://www.mobiletransaction.org>
- 118 **[PCMCIA]** Personal Computer Memory Card International Association. *PC Card Standard,*
119 *Release 2.1.* July 1993.
- 120 **[PKCS #1]** RSA Laboratories. *RSA Cryptography Standard, v2.1.* June 14, 2002 URL:
121 <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>
- 122 **[PKCS #3]** RSA Laboratories. *Diffie-Hellman Key-Agreement Standard, v1.4.* November
123 1993.
- 124 **[PKCS #5]** RSA Laboratories. *Password-Based Encryption Standard, v2.0.* March 26,
125 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf>
- 126 **[PKCS #7]** RSA Laboratories. *Cryptographic Message Syntax Standard, v1.6.* November
127 1997 URL : <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v16.pdf>
- 128 **[PKCS #8]** RSA Laboratories. *Private-Key Information Syntax Standard, v1.2.* November
129 1993. URL : ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn
- 130 **[PKCS #11-UG]** *PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40.* Edited by
131 John Leiseboer and Robert Griffin. Latest version. [http://docs.oasis-](http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html)
132 [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).
- 133 **[PKCS #12]** RSA Laboratories. *Personal Information Exchange Syntax Standard, v1.0.*
134 June 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf>
- 135 **[RFC 1321]** R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* MIT Laboratory for
136 Computer Science and RSA Data Security, Inc., April 1992. URL:
137 <http://www.rfc-editor.org/rfc/rfc1321.txt>

138 **[RFC 3369]** R. Houseley. *RFC 3369: Cryptographic Message Syntax (CMS)*. August 2002.
139 URL: <http://www.rfc-editor.org/rfc/rfc3369.txt>

140 **[RFC 6149]** S. Turner and L. Chen. *RFC 6149: MD2 to Historic Status*. March, 2011. URL:
141 <http://www.rfc-editor.org/rfc/rfc6149.txt>

142 **[SEC-1]** Standards for Efficient Cryptography Group (SECG). *Standards for Efficient
143 Cryptography (SEC) 1: Elliptic Curve Cryptography*. Version 1.0, September 20,
144 2000.

145 **[SEC-2]** Standards for Efficient cryptography Group (SECG). Standards for Efficient
146 Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters.
147 Version 1.0, September 20, 2000.

148 **[TLS]** IETF. *RFC 2246: The TLS Protocol Version 1.0*. January 1999. URL:
149 <http://ietf.org/rfc/rfc2256.txt>

150 **[WIM]** WAP. *Wireless Identity Module. – WAP-260-WIP-20010712.a*. July 2001. URL:
151 [http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?Doc
152 Name=/wap/wap-260-wim-20010712-a.pdf](http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf)

153 **[WPKI]** WAP. *Wireless Application Protocol: Public Key Infrastructure Definition. – WAP-
154 217-WPKI-20010424-a*. April 2001. URL:
155 [http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?Doc
156 Name=/wap/wap-217-wpki-20010424-a.pdf](http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf)

157 **[WTLS]** WAP. *Wireless Transport Layer Security Version – WAP-261-WTLS-20010406-
158 a*. April 2001. URL:
159 [http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?Doc
160 Name=/wap/wap-261-wtls-20010406-a.pdf](http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf)

161 **[X.500]** ITU-T. Information Technology – Open Systems Interconnection –The Directory:
162 Overview of Concepts, Models and Services. February 2001. (Identical to
163 ISO/IEC 9594-1)

164 **[X.509]** ITU-T. Information Technology – Open Systems Interconnection – The
165 Directory: Public-key and Attribute Certificate Frameworks. March 2000.
166 (Identical to ISO/IEC 9594-8)

167 **[X.680]** ITU-T. Information Technology – Abstract Syntax Notation One (ASN.1):
168 Specification of Basic Notation. July 2002. (Identical to ISO/IEC 8824-1)

169 **[X.690]** ITU-T. Information Technology – ASN.1 Encoding Rules: Specification of Basic
170 Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished
171 Encoding Rules (DER). July 2002. (Identical to ISO/IEC 8825-1)

172 **2 Mechanisms**

173 **2.1 PKCS #11 Mechanisms**

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

176
 177 The following table shows which Cryptoki mechanisms are supported by different cryptographic
 178 operations. For any particular token, of course, a particular operation MAY support only a subset of the
 179 mechanisms listed. There is also no guarantee that a token which supports one mechanism for some
 180 operation supports any other mechanism for any other operation (or even supports that same mechanism
 181 for any other operation). For example, even if a token is able to create RSA digital signatures with the
 182 **CKM_RSA_PKCS** mechanism, it may or may not be the case that the same token MAY also perform
 183 RSA encryption with **CKM_RSA_PKCS**.

184 *Table 1, Mechanisms vs. Functions*

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_FORTEZZA_TIMESTAMP		X ²					
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 ¹	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_OFB64	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 ³	
CKM_BATON_KEY_GEN					X		
CKM_BATON_ECB128	X						
CKM_BATON_ECB96	X						

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	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	

185 ¹ SR = SignRecover, VR = VerifyRecover.

186 ² Single-part operations only.

187 ³ Mechanism MUST only be used for wrapping, not unwrapping.

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

190 In general, if a mechanism makes no mention of the *ulMinKeyLen* and *ulMaxKeyLen* fields of the
191 CK_MECHANISM_INFO structure, then those fields have no meaning for that particular mechanism.
192

193 2.2 FORTEZZA timestamp

194 The FORTEZZA timestamp mechanism, denoted **CKM_FORTEZZA_TIMESTAMP**, is a mechanism for
195 single-part signatures and verification. The signatures it produces and verifies are DSA digital signatures
196 over the provided hash value and the current time.

197 **It has no parameters.**

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

200 *Table 2, FORTEZZA Timestamp: Key and Data Length*

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

201 ¹ Single-part operations only

202 ² Data length, signature length

203 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
204 specify the supported range of DSA prime sizes, in bits.

205 2.3 KEA

206 2.3.1 Definitions

207 This section defines the key type “CKK_KEA” for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
208 attribute of key objects.

209 Mechanisms:

210 CKM_KEA_KEY_PAIR_GEN

211 CKM_KEA_KEY_DERIVE

212 2.3.2 KEA mechanism parameters

213 2.3.2.1 CK_KEA_DERIVE_PARAMS; CK_KEA_DERIVE_PARAMS_PTR

214

215 **CK_KEA_DERIVE_PARAMS** is a structure that provides the parameters to the **CKM_KEA_DERIVE**
216 mechanism. It is defined as follows:

```
217 typedef struct CK_KEA_DERIVE_PARAMS {  
218     CK_BBOOL isSender;  
219     CK_ULONG ulRandomLen;  
220     CK_BYTE_PTR pRandomA;  
221     CK_BYTE_PTR pRandomB;  
222     CK_ULONG ulPublicDataLen;  
223     CK_BYTE_PTR pPublicData;  
224 } CK_KEA_DERIVE_PARAMS;
```

225

226 The fields of the structure have the following meanings:

227 *isSender* Option for generating the key (called a TEK). The value
 228 is CK_TRUE if the sender (originator) generates the
 229 TEK, CK_FALSE if the recipient is regenerating the TEK

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

231 *pRandomA* pointer to Ra data

232 *pRandomB* pointer to Rb data

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

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

235 **CK_KEA_DERIVE_PARAMS_PTR** is a pointer to a **CK_KEA_DERIVE_PARAMS**.

236 2.3.3 KEA public key objects

237 KEA public key objects (object class **CKO_PUBLIC_KEY**, key type **CKK_KEA**) hold KEA public keys.
 238 The following table defines the KEA public key object attributes, in addition to the common attributes
 239 defined for this object class:

240 *Table 3, KEA Public Key Object Attributes*

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

241 Refer to [PKCS #11-Base] table 10 for footnotes

242 The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the "KEA domain
 243 parameters".

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

```

245 CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
246 CK_KEY_TYPE keyType = CKK_KEA;
247 CK_UTF8CHAR label[] = "A KEA public key object";
248 CK_BYTE prime[] = {...};
249 CK_BYTE subprime[] = {...};
250 CK_BYTE base[] = {...};
251 CK_BYTE value[] = {...};
252 CK_ATTRIBUTE template[] = {
253     {CKA_CLASS, &class, sizeof(class)},
254     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
255     {CKA_TOKEN, &>true, sizeof(true)},
256     {CKA_LABEL, label, sizeof(label)-1},
257     {CKA_PRIME, prime, sizeof(prime)},
258     {CKA_SUBPRIME, subprime, sizeof(subprime)},
259     {CKA_BASE, base, sizeof(base)},
260     {CKA_VALUE, value, sizeof(value)}
261 };
  
```

262

263 **2.3.4 KEA private key objects**

264 KEA private key objects (object class **CKO_PRIVATE_KEY**, key type **CKK_KEA**) hold KEA private keys.
265 The following table defines the KEA private key object attributes, in addition to the common attributes
266 defined for this object class:

267 *Table 4, KEA Private Key Object Attributes*

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

268 Refer to [PKCS #11-Base] table 10 for footnotes

269
270 The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the “KEA domain
271 parameters”.

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

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

```

276 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
277 CK_KEY_TYPE keyType = CKK_KEA;
278 CK_UTF8CHAR label[] = "A KEA private key object";
279 CK_BYTE subject[] = {...};
280 CK_BYTE id[] = {123};
281 CK_BYTE prime[] = {...};
282 CK_BYTE subprime[] = {...};
283 CK_BYTE base[] = {...};
284 CK_BYTE value[] = {...};
285 CK_BBOOL true = CK_TRUE;
286 CK_ATTRIBUTE template[] = {
287     {CKA_CLASS, &class, sizeof(class)},
288     {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS
289     {CKA_TOKEN, &>true, sizeof(true)},
290     {CKA_LABEL, label, sizeof(label) - 1},
291     {CKA_SUBJECT, subject, sizeof(subject)},
292     {CKA_ID, id, sizeof(id)},
293     {CKA_SENSITIVE, &>true, sizeof(true)},
294     {CKA_DERIVE, &>true, sizeof(true)},
295     {CKA_PRIME, prime, sizeof(prime)},
296     {CKA_SUBPRIME, subprime, sizeof(subprime)},
297     {CKA_BASE, base, sizeof(base)},
298     {CKA_VALUE, value, sizeof(value)}
299 };

```

300 **2.3.5 KEA key pair generation**

301 The KEA key pair generation mechanism, denoted **CKM_KEA_KEY_PAIR_GEN**, generates key pairs for
302 the Key Exchange Algorithm, as defined by NIST’s “SKIPJACK and KEA Algorithm Specification Version
303 2.0”, 29 May 1998.

304 It does not have a parameter.

305 The mechanism generates KEA public/private key pairs with a particular prime, subprime and base, as
306 specified in the **CKA_PRIME**, **CKA_SUBPRIME**, and **CKA_BASE** attributes of the template for the public

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

309 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE** and **CKA_VALUE** attributes to the new
 310 public key and the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**, **CKA_BASE**, and
 311 **CKA_VALUE** attributes to the new private key. Other attributes supported by the KEA public and private
 312 key types (specifically, the flags indicating which functions the keys support) MAY also be specified in the
 313 templates for the keys, or else are assigned default initial values.

314 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 315 specify the supported range of KEA prime sizes, in bits.

316 2.3.6 KEA key derivation

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

320 It has a parameter, a **CK_KEA_DERIVE_PARAMS** structure.

321 This mechanism derives a secret value, and truncates the result according to the **CKA_KEY_TYPE**
 322 attribute of the template and, if it has one and the key type supports it, the **CKA_VALUE_LEN** attribute of
 323 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism
 324 contributes the result as the **CKA_VALUE** attribute of the new key; other attributes required by the key
 325 type must be specified in the template.

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

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

336 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.

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

339 key. In these cases, any object template supplied as the **C_DeriveKey** *pTemplate* argument should be
340 ignored.

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

- 342 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key MAY
343 both be specified to be either CK_TRUE or CK_FALSE. If omitted, these attributes each take on
344 some default value.
- 345 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, then the derived
346 key MUST as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to
347 CK_TRUE, then the derived has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value
348 as its **CKA_SENSITIVE** attribute.
- 349 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE, then
350 the derived key MUST, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set
351 to CK_TRUE, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the
352 *opposite* value from its **CKA_EXTRACTABLE** attribute.

353 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
354 specify the supported range of KEA prime sizes, in bits.

355 2.4 RC2

356 2.4.1 Definitions

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

362 This section defines the key type “CKK_RC2” for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
363 attribute of key objects.

364 Mechanisms:

- 365 CKM_RC2_KEY_GEN
- 366 CKM_RC2_ECB
- 367 CKM_RC2_CBC
- 368 CKM_RC2_MAC
- 369 CKM_RC2_MAC_GENERAL
- 370 CKM_RC2_CBC_PAD

371 2.4.2 RC2 secret key objects

372 RC2 secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_RC2**) hold RC2 keys. The
373 following table defines the RC2 secret key object attributes, in addition to the common attributes defined
374 for this object class:

375 *Table 6, RC2 Secret Key Object Attributes*

* 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.

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (1 to 128 bytes)
CKA_VALUE_LEN ^{2,3}	CK_ULONG	Length in bytes of key value

376 Refer to [PKCS #11-Base] table 10 for footnotes

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

```

378 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
379 CK_KEY_TYPE keyType = CKK_RC2;
380 CK_UTF8CHAR label[] = "An RC2 secret key object";
381 CK_BYTE value[] = {...};
382 CK_BBOOL true = CK_TRUE;
383 CK_ATTRIBUTE template[] = {
384     {CKA_CLASS, &class, sizeof(class)},
385     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
386     {CKA_TOKEN, &true, sizeof(true)},
387     {CKA_LABEL, label, sizeof(label)-1},
388     {CKA_ENCRYPT, &true, sizeof(true)},
389     {CKA_VALUE, value, sizeof(value)}
390 };

```

391 2.4.3 RC2 mechanism parameters

392 2.4.3.1 CK_RC2_PARAMS; CK_RC2_PARAMS_PTR

393 **CK_RC2_PARAMS** provides the parameters to the **CKM_RC2_ECB** and **CKM_RC2_MAC** mechanisms.
394 It holds the effective number of bits in the RC2 search space. It is defined as follows:

```

395 typedef CK_ULONG CK_RC2_PARAMS;

```

396 **CK_RC2_PARAMS_PTR** is a pointer to a **CK_RC2_PARAMS**.

397 2.4.3.2 CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR

398 **CK_RC2_CBC_PARAMS** is a structure that provides the parameters to the **CKM_RC2_CBC** and
399 **CKM_RC2_CBC_PAD** mechanisms. It is defined as follows:

```

400 typedef struct CK_RC2_CBC_PARAMS {
401     CK_ULONG ulEffectiveBits;
402     CK_BYTE iv[8];
403 } CK_RC2_CBC_PARAMS;

```

404 The fields of the structure have the following meanings:

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

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

408 **CK_RC2_CBC_PARAMS_PTR** is a pointer to a **CK_RC2_CBC_PARAMS**.

409 2.4.3.3 CK_RC2_MAC_GENERAL_PARAMS; 410 CK_RC2_MAC_GENERAL_PARAMS_PTR

411 **CK_RC2_MAC_GENERAL_PARAMS** is a structure that provides the parameters to the
412 **CKM_RC2_MAC_GENERAL** mechanism. It is defined as follows:

```

413 typedef struct CK_RC2_MAC_GENERAL_PARAMS {
414     CK_ULONG ulEffectiveBits;

```

```

415     CK_ULONG ulMacLength;
416 } CK_RC2_MAC_GENERAL_PARAMS;

```

417 The fields of the structure have the following meanings:

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

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

420 **CK_RC2_MAC_GENERAL_PARAMS_PTR** is a pointer to a **CK_RC2_MAC_GENERAL_PARAMS**.

421 2.4.4 RC2 key generation

422 The RC2 key generation mechanism, denoted **CKM_RC2_KEY_GEN**, is a key generation mechanism for
 423 RSA Security's block cipher RC2.

424 It does not have a parameter.

425 The mechanism generates RC2 keys with a particular length in bytes, as specified in the
 426 **CKA_VALUE_LEN** attribute of the template for the key.

427 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
 428 key. Other attributes supported by the RC2 key type (specifically, the flags indicating which functions the
 429 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

430 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 431 specify the supported range of RC2 key sizes, in bits.

432 2.4.5 RC2-ECB

433 RC2-ECB, denoted **CKM_RC2_ECB**, is a mechanism for single- and multiple-part encryption and
 434 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and electronic
 435 codebook mode as defined in FIPS PUB 81.

436 It has a parameter, a **CK_RC2_PARAMS**, which indicates the effective number of bits in the RC2 search
 437 space.

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

444 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
 445 **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the
 446 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
 447 attribute of the new key; other attributes required by the key type must be specified in the template.

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

449 *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	Determined by type of key being unwrapped or	

		8	CKA_VALUE_LEN	
--	--	---	---------------	--

450 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
451 specify the supported range of RC2 effective number of bits.

452 2.4.6 RC2-CBC

453 RC2_CBC, denoted **CKM_RC2_CBC**, is a mechanism for single- and multiple-part encryption and
454 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and cipher-
455 block chaining mode as defined in FIPS PUB 81.

456 It has a parameter, a **CK_RC2_CBC_PARAMS** structure, where the first field indicates the effective
457 number of bits in the RC2 search space, and the next field is the initialization vector for cipher block
458 chaining mode.

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

465 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
466 **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the
467 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
468 attribute of the new key; other attributes required by the key type must be specified in the template.

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

470 *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	

471 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
472 specify the supported range of RC2 effective number of bits.

473 2.4.7 RC2-CBC with PKCS padding

474 RC2-CBC with PKCS padding, denoted **CKM_RC2_CBC_PAD**, is a mechanism for single- and multiple-
475 part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher
476 RC2; cipher-block chaining mode as defined in FIPS PUB 81; and the block cipher padding method
477 detailed in PKCS #7.

478 It has a parameter, a **CK_RC2_CBC_PARAMS** structure, where the first field indicates the effective
479 number of bits in the RC2 search space, and the next field is the initialization vector.

480 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the
481 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
482 for the **CKA_VALUE_LEN** attribute.

483 In addition to being able to wrap and unwrap secret keys, this mechanism MAY wrap and unwrap RSA,
484 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see **[PKCS #11-**

485 **Curr], Miscellaneous simple key derivation mechanisms** for details). The entries in the table below
 486 for data length constraints when wrapping and unwrapping keys do not apply to wrapping and
 487 unwrapping private keys.

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

489 *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

490 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 491 specify the supported range of RC2 effective number of bits.

492 2.4.8 General-length RC2-MAC

493 General-length RC2-MAC, denoted **CKM_RC2_MAC_GENERAL**, is a mechanism for single-and
 494 multiple-part signatures and verification, based on RSA Security's block cipher RC2 and data
 495 authorization as defined in FIPS PUB 113.

496 It has a parameter, a **CK_RC2_MAC_GENERAL_PARAMS** structure, which specifies the effective
 497 number of bits in the RC2 search space and the output length desired from the mechanism.

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

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

501 *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

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 2.4.9 RC2-MAC

505 RC2-MAC, denoted by **CKM_RC2_MAC**, is a special case of the general-length RC2-MA mechanism
 506 (see Section 2.4.8). Instead of taking a **CK_RC2_MAC_GENERAL_PARAMS** parameter, it takes a
 507 **CK_RC2_PARAMS** parameter, which only contains the effective number of bits in the RC2 search space.
 508 RC2-MAC produces and verifies 4-byte MACs.

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

510

511 *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

512 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
513 specify the supported range of RC2 effective number of bits.

514 2.5 RC4

515 2.5.1 Definitions

516 This section defines the key type “CKK_RC4” for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
517 attribute of key objects.

518 Mechanisms

519 CKM_RC4_KEY_GEN

520 CKM_RC4

521 2.5.2 RC4 secret key objects

522 RC4 secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_RC4**) hold RC4 keys. The
523 following table defines the RC4 secret key object attributes, in addition to the common attributes defined
524 for this object class:

525 *Table 12, RC4 Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (1 to 256 bytes)
CKA_VALUE_LEN ^{2,3,6}	CK_ULONG	Length in bytes of key value

526 Refer to [PKCS #11-Base] table 10 for footnotes

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

```
528 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
529 CK_KEY_TYPE keyType = CKK_RC4;  
530 CK_UTF8CHAR label[] = "An RC4 secret key object";  
531 CK_BYTE value[] = {...};  
532 CK_BBOOL true = CK_TRUE;  
533 CK_ATTRIBUTE template[] = {  
534     {CKA_CLASS, &class, sizeof(class)},  
535     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
536     {CKA_TOKEN, &>true, sizeof(true)},  
537     {CKA_LABEL, label, sizeof(label)-1},  
538     {CKA_ENCRYPT, &>true, sizeof(true)},  
539     {CKA_VALUE, value, sizeof(value)}  
540 };
```

541 2.5.3 RC4 key generation

542 The RC4 key generation mechanism, denoted **CKM_RC4_KEY_GEN**, is a key generation mechanism for
543 RSA Security’s proprietary stream cipher RC4.

544 It does not have a parameter.

545 The mechanism generates RC4 keys with a particular length in bytes, as specified in the
546 **CKA_VALUE_LEN** attribute of the template for the key.

547 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
548 key. Other attributes supported by the RC4 key type (specifically, the flags indicating which functions the
549 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

550 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
551 specify the supported range of RC4 key sizes, in bits.

552 2.5.4 RC4 mechanism

553 RC4, denoted **CKM_RC4**, is a mechanism for single- and multiple-part encryption and decryption based
554 on RSA Security's proprietary stream cipher RC4.

555 It does not have a parameter.

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

557 *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

558 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
559 specify the supported range of RC4 key sizes, in bits.

560 2.6 RC5

561 2.6.1 Definitions

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

564 This section defines the key type "CKK_RC5" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
565 attribute of key objects.

566 Mechanisms:

567 CKM_RC5_KEY_GEN

568 CKM_RC5_ECB

569 CKM_RC5_CBC

570 CKM_RC5_MAC

571 CKM_RC5_MAC_GENERAL

572 CMK_RC5_CBC_PAD

573 2.6.2 RC5 secret key objects

574 RC5 secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_RC5**) hold RC5 keys. The
575 following table defines the RC5 secret key object attributes, in addition to the common attributes defined
576 for this object class.

577 *Table 14, RC5 Secret Key Object*

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

578 Refer to [PKCS #11-Base] table 10 for footnotes

579

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

```
581 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
582 CK_KEY_TYPE keyType = CKK_RC5;  
583 CK_UTF8CHAR label[] = "An RC5 secret key object";  
584 CK_BYTE value[] = {...};  
585 CK_BBOOL true = CK_TRUE;
```

```

586 CK_ATTRIBUTE template[] = {
587     {CKA_CLASS, &class, sizeof(class)},
588     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
589     {CKA_TOKEN, &true, sizeof(true)},
590     {CKA_LABEL, label, sizeof(label)-1},
591     {CKA_ENCRYPT, &true, sizeof(true)},
592     {CKA_VALUE, value, sizeof(value)}
593 };

```

594 2.6.3 RC5 mechanism parameters

595 2.6.3.1 CK_RC5_PARAMS; CK_RC5_PARAMS_PTR

596 **CK_RC5_PARAMS** provides the parameters to the **CKM_RC5_ECB** and **CKM_RC5_MAC** mechanisms.
597 It is defined as follows:

```

598 typedef struct CK_RC5_PARAMS {
599     CK_ULONG ulWordsize;
600     CK_ULONG ulRounds;
601 } CK_RC5_PARAMS;

```

602 The fields of the structure have the following meanings:

603 *ulWordsize* wordsize of RC5 cipher in bytes

604 *ulRounds* number of rounds of RC5 encipherment

605 **CK_RC5_PARAMS_PTR** is a pointer to a **CK_RC5_PARAMS**.

606 2.6.3.2 CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR

607 **CK_RC5_CBC_PARAMS** is a structure that provides the parameters to the **CKM_RC5_CBC** and
608 **CKM_RC5_CBC_PAD** mechanisms. It is defined as follows:

```

609 typedef struct CK_RC5_CBC_PARAMS {
610     CK_ULONG ulWordsize;
611     CK_ULONG ulRounds;
612     CK_BYTE_PTR pIv;
613     CK_ULONG ulIvLen;
614 } CK_RC5_CBC_PARAMS;

```

615 The fields of the structure have the following meanings:

616 *ulwordSize* wordsize of RC5 cipher in bytes

617 *ulRounds* number of rounds of RC5 encipherment

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

619 *ulIvLen* length of initialization vector (must be same as
620 blocksize)

621 **CK_RC5_CBC_PARAMS_PTR** is a pointer to a **CK_RC5_CBC_PARAMS**.

622 2.6.3.3 CK_RC5_MAC_GENERAL_PARAMS; 623 CK_RC5_MAC_GENERAL_PARAMS_PTR

624 **CK_RC5_MAC_GENERAL_PARAMS** is a structure that provides the parameters to the
625 **CKM_RC5_MAC_GENERAL** mechanism. It is defined as follows:

```

626 typedef struct CK_RC5_MAC_GENERAL_PARAMS {
627     CK_ULONG ulWordsize;
628     CK_ULONG ulRounds;
629     CK_ULONG ulMacLength;
630 } CK_RC5_MAC_GENERAL_PARAMS;

```

631 The fields of the structure have the following meanings:

- 632 *ulwordSize* wordsize of RC5 cipher in bytes
- 633 *ulRounds* number of rounds of RC5 encipherment
- 634 *ulMacLength* length of the MAC produced, in bytes

635 **CK_RC5_MAC_GENERAL_PARAMS_PTR** is a pointer to a **CK_RC5_MAC_GENERAL_PARAMS**.

636 2.6.4 RC5 key generation

637 The RC5 key generation mechanism, denoted **CKM_RC5_KEY_GEN**, is a key generation mechanism for
638 RSA Security's block cipher RC5.

639 It does not have a parameter.

640 The mechanism generates RC5 keys with a particular length in bytes, as specified in the
641 **CKA_VALUE_LEN** attribute of the template for the key.

642 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
643 key. Other attributes supported by the RC5 key type (specifically, the flags indicating which functions the
644 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

645 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
646 specify the supported range of RC5 key sizes, in bytes.

647 2.6.5 RC5-ECB

648 RC5-ECB, denoted **CKM_RC5_ECB**, is a mechanism for single- and multiple-part encryption and
649 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and electronic
650 codebook mode as defined in FIPS PUB 81.

651 It has a parameter, **CK_RC5_PARAMS**, which indicates the wordsize and number of rounds of
652 encryption to use.

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

659 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
660 **CKA_KEY_TYPE** attributes of the template and, if it has one, and the key type supports it, the
661 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
662 attribute of the new key; other attributes required by the key type must be specified in the template.

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

664 *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	

665 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
666 specify the supported range of RC5 key sizes, in bytes.

667 2.6.6 RC5-CBC

668 RC5-CBC, denoted **CKM_RC5_CBC**, is a mechanism for single- and multiple-part encryption and
669 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and cipher-
670 block chaining mode as defined in FIPS PUB 81.

671 It has a parameter, a **CK_RC5_CBC_PARAMS** structure, which specifies the wordsize and number of
672 rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.

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

679 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
680 **CKA_KEY_TYPE** attribute for the template, and, if it has one, and the key type supports it, the
681 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
682 attribute of the new key; other attributes required by the key type must be specified in the template.

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

684 *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	

685 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
686 specify the supported range of RC5 key sizes, in bytes.

687 2.6.7 RC5-CBC with PKCS padding

688 RC5-CBC with PKCS padding, denoted **CKM_RC5_CBC_PAD**, is a mechanism for single- and multiple-
689 part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher
690 RC5; cipher block chaining mode as defined in FIPS PUB 81; and the block cipher padding method
691 detailed in PKCS #7.

692 It has a parameter, a **CK_RC5_CBC_PARAMS** structure, which specifies the wordsize and number of
693 rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.

694 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the
695 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
696 for the **CKA_VALUE_LEN** attribute.

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

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

702 *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

703 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
704 specify the supported range of RC5 key sizes, in bytes.

705 **2.6.8 General-length RC5-MAC**

706 General-length RC5-MAC, denoted **CKM_RC5_MAC_GENERAL**, is a mechanism for single- and
707 multiple-part signatures and verification, based on RSA Security's block cipher RC5 and data
708 authentication as defined in FIPS PUB 113.

709 It has a parameter, a **CK_RC5_MAC_GENERAL_PARAMS** structure, which specifies the wordsize and
710 number of rounds of encryption to use and the output length desired from the mechanism.

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

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

714 *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

715 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
716 specify the supported range of RC5 key sizes, in bytes.

717 **2.6.9 RC5-MAC**

718 RC5-MAC, denoted by **CKM_RC5_MAC**, is a special case of the general-length RC5-MAC mechanism.
719 Instead of taking a **CK_RC5_MAC_GENERAL_PARAMS** parameter, it takes a **CK_RC5_PARAMS**
720 parameter. RC5-MAC produces and verifies MACs half as large as the RC5 blocksize.

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

722 *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]

723 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
724 specify the supported range of RC5 key sizes, in bytes.

725 2.7 General block cipher

726 2.7.1 Definitions

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

730 This section defines the key types "CKK_DES", "CKK_CAST", "CKK_CAST3", "CKK_CAST5"
731 (deprecated in v2.11), "CKK_CAST128", "CKK_IDEA" and "CKK_CDMF" for type CK_KEY_TYPE as
732 used in the CKA_KEY_TYPE attribute of key objects.

733 Mechanisms:

734 CKM_DES_KEY_GEN
735 CKM_DES_ECB
736 CKM_DES_CBC
737 CKM_DES_MAC
738 CKM_DES_MAC_GENERAL
739 CKM_DES_CBC_PAD
740 CKM_CDMF_KEY_GEN
741 CKM_CDMF_ECB
742 CKM_CDMF_CBC
743 CKM_CDMF_MAC
744 CKM_CDMF_MAC_GENERAL
745 CKM_CDMF_CBC_PAD
746 CKM_DES_OFB64
747 CKM_DES_OFB8
748 CKM_DES_CFB64
749 CKM_DES_CFB8
750 CKM_CAST_KEY_GEN
751 CKM_CAST_ECB
752 CKM_CAST_CBC
753 CKM_CAST_MAC
754 CKM_CAST_MAC_GENERAL
755 CKM_CAST_CBC_PAD
756 CKM_CAST3_KEY_GEN
757 CKM_CAST3_ECB
758 CKM_CAST3_CBC
759 CKM_CAST3_MAC
760 CKM_CAST3_MAC_GENERAL

761 CKM_CAST3_CBC_PAD
 762 CKM_CAST5_KEY_GEN
 763 CKM_CAST128_KEY_GEN
 764 CKM_CAST5_ECB
 765 CKM_CAST128_ECB
 766 CKM_CAST5_CBC
 767 CKM_CAST128_CBC_CBC
 768 CKM_CAST5_MAC
 769 CKM_CAST128_MAC
 770 CKM_CAST5_MAC_GENERAL
 771 CKM_CAST128_MAC_GENERAL
 772 CKM_CAST5_CBC_PAD
 773 CKM_CAST128_CBC_PAD
 774 CKM_IDEA_KEY_GEN
 775 CKM_IDEA_ECB
 776 CKM_IDEA_MAC
 777 CKM_IDEA_MAC_GENERAL
 778 CKM_IDEA_CBC_PAD

779 **2.7.2 DES secret key objects**

780 DES secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_DES**) hold single-length DES
 781 keys. The following table defines the DES secret key object attributes, in addition to the common
 782 attributes defined for this object class:

783 *Table 20, DES Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (8 bytes long)

784 Refer to [PKCS #11-Base] table 10 for footnotes

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

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

```

788 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
789 CK_KEY_TYPE keyType = CKK_DES;
790 CK_UTF8CHAR label[] = "A DES secret key object";
791 CK_BYTE value[8] = {...};
792 CK_BBOOL true = CK_TRUE;
793 CK_ATTRIBUTE template[] = {
794     {CKA_CLASS, &class, sizeof(class)},
795     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
796     {CKA_TOKEN, &>true, sizeof(true)},
797     {CKA_LABEL, label, sizeof(label)-1},
798     {CKA_ENCRYPT, &>true, sizeof(true)},
799     {CKA_VALUE, value, sizeof(value)}
800 };
  
```

801 CKA_CHECK_VALUE: The value of this attribute is derived from the key object by taking the first three
 802 bytes of the ECB encryption of a single block of null (0x00) bytes, using the default cipher associated with
 803 the key type of the secret key object.

804 **2.7.3 CAST secret key objects**

805 CAST secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_CAST**) hold CAST keys.
806 The following table defines the CAST secret key object attributes, in addition to the common attributes
807 defined for this object class:

808 *Table 21, CAST Secret Key Object Attributes*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (1 to 8 bytes)
CKA_VALUE_LEN ^{2,3,6}	CK_ULONG	Length in bytes of key value

809 Refer to [PKCS #11-Base] table 10 for footnotes

810

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

```

812 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
813 CK_KEY_TYPE keyType = CKK_CAST;
814 CK_UTF8CHAR label[] = "A CAST secret key object";
815 CK_BYTE value[] = {...};
816 CK_BBOOL true = CK_TRUE;
817 CK_ATTRIBUTE template[] = {
818     {CKA_CLASS, &class, sizeof(class)},
819     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
820     {CKA_TOKEN, &>true, sizeof(true)},
821     {CKA_LABEL, label, sizeof(label)-1},
822     {CKA_ENCRYPT, &>true, sizeof(true)},
823     {CKA_VALUE, value, sizeof(value)}
824 };

```

825 **2.7.4 CAST3 secret key objects**

826 CAST3 secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_CAST3**) hold CAST3 keys.
827 The following table defines the CAST3 secret key object attributes, in addition to the common attributes
828 defines for this object class:

829 *Table 22, CAST3 Secret Key Object Attributes*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (1 to 8 bytes)
CKA_VALUE_LEN ^{2,3,6}	CK_ULONG	Length in bytes of key value

830 Refer to [PKCS #11-Base] table 10 for footnotes

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

```

832 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
833 CK_KEY_TYPE keyType = CKK_CAST3;
834 CK_UTF8CHAR label[] = "A CAST3 secret key object";
835 CK_BYTE value[] = {...};
836 CK_BBOOL true = CK_TRUE;
837 CK_ATTRIBUTE template[] = {
838     {CKA_CLASS, &class, sizeof(class)},
839     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
840     {CKA_TOKEN, &>true, sizeof(true)},
841     {CKA_LABEL, label, sizeof(label)-1},
842     {CKA_ENCRYPT, &>true, sizeof(true)},
843     {CKA_VALUE, value, sizeof(value)}
844 };

```

845 2.7.5 CAST128 (CAST5) secret key objects

846 CAST128 (also known as CAST5) secret key objects (object class **CKO_SECRET_KEY**, key type
847 **CKK_CAST128** or **CKK_CAST5**) hold CAST128 keys. The following table defines the CAST128 secret
848 key object attributes, in addition to the common attributes defines for this object class:

849 Table 23, CAST128 (CAST5) Secret Key Object Attributes

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (1 to 16 bytes)
CKA_VALUE_LEN ^{2,3,6}	CK_ULONG	Length in bytes of key value

850 Refer to [PKCS #11-Base] table 10 for footnotes

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

```
852 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
853 CK_KEY_TYPE keyType = CKK_CAST128;  
854 CK_UTF8CHAR label[] = "A CAST128 secret key object";  
855 CK_BYTE value[] = {...};  
856 CK_BBOOL true = CK_TRUE;  
857 CK_ATTRIBUTE template[] = {  
858     {CKA_CLASS, &class, sizeof(class)},  
859     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
860     {CKA_TOKEN, &>true, sizeof(true)},  
861     {CKA_LABEL, label, sizeof(label)-1},  
862     {CKA_ENCRYPT, &>true, sizeof(true)},  
863     {CKA_VALUE, value, sizeof(value)}  
864 };
```

865

866 2.7.6 IDEA secret key objects

867 IDEA secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_IDEA**) hold IDEA keys. The following
868 table defines the IDEA secret key object attributes, in addition to the common attributes defines for this object class:

869 Table 24, IDEA Secret Key Object

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (16 bytes long)

870 Refer to [PKCS #11-Base] table 10 for footnotes

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

```
872 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
873 CK_KEY_TYPE keyType = CKK_IDEA;  
874 CK_UTF8CHAR label[] = "An IDEA secret key object";  
875 CK_BYTE value[16] = {...};  
876 CK_BBOOL true = CK_TRUE;  
877 CK_ATTRIBUTE template[] = {  
878     {CKA_CLASS, &class, sizeof(class)},  
879     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
880     {CKA_TOKEN, &>true, sizeof(true)},  
881     {CKA_LABEL, label, sizeof(label)-1},  
882     {CKA_ENCRYPT, &>true, sizeof(true)},  
883     {CKA_VALUE, value, sizeof(value)}  
884 };
```

885

886 2.7.7 CDMF secret key objects

887 *IDEA secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_CDMF**) hold CDMF keys. The following*
888 *table defines the CDMF secret key object attributes, in addition to the common attributes defines for this object class:*

889 *Table 25, CDMF Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (8 bytes long)

890 Refer to [PKCS #11-Base] table 10 for footnotes

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

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

```
895 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
896 CK_KEY_TYPE keyType = CKK_CDMF;  
897 CK_UTF8CHAR label[] = "A CDMF secret key object";  
898 CK_BYTE value[8] = {...};  
899 CK_BBOOL true = CK_TRUE;  
900 CK_ATTRIBUTE template[] = {  
901     {CKA_CLASS, &class, sizeof(class)},  
902     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
903     {CKA_TOKEN, &>true, sizeof(true)},  
904     {CKA_LABEL, label, sizeof(label)-1},  
905     {CKA_ENCRYPT, &>true, sizeof(true)},  
906     {CKA_VALUE, value, sizeof(value)}  
907 };
```

908 2.7.8 General block cipher mechanism parameters

909 2.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR

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

```
916 typedef CK_ULONG CK_MAC_GENERAL_PARAMS;  
917
```

918 **CK_MAC_GENERAL_PARAMS_PTR** is a pointer to a **CK_MAC_GENERAL_PARAMS**.

919 2.7.9 General block cipher key generation

920 Cipher <NAME> has a key generation mechanism, "<NAME> key generation", denoted by
921 **CKM_<NAME>_KEY_GEN**.

922 This mechanism does not have a parameter.

923 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
924 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key
925 supports) MAY be specified in the template for the key, or else are assigned default initial values.

926 When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB
927 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits
928 set properly.

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

932 When CAST, CAST3, or CAST128 (CAST5) keys are generated, the template for the secret key must
 933 specify a **CKA_VALUE_LEN** attribute.

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

939 2.7.10 General block cipher ECB

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

943 It does not have a parameter.

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

950 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
 951 **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the
 952 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
 953 attribute of the new key; other attributes required by the key must be specified in the template.

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

955 *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	

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

961 2.7.11 General block cipher CBC

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

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

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

968 *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	

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

974 2.7.12 General block cipher CBC with PKCS padding

975 Cipher <NAME> has a cipher-block chaining mode with PKCS padding, “<NAME>-CBC with PKCS
 976 padding”, denoted **CKM_<NAME>_CBC_PAD**. It is a mechanism for single- and multiple-part encryption
 977 and decryption; key wrapping; and key unwrapping with <NAME>. All ciphertext is padded with PKCS
 978 padding.

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

981 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the
 982 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
 983 for the **CKA_VALUE_LEN** attribute.

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

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

989 *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 blocksize	Between 1 and blocksize bytes shorter than input length

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

995 **2.7.13 General-length general block cipher MAC**

996 Cipher <NAME> has a general-length MACing mode, “General-length <NAME>-MAC”, denoted
 997 **CKM_<NAME>_MAC_GENERAL**. It is a mechanism for single-and multiple-part signatures and
 998 verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB
 999 113.

1000 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which specifies the size of the output.

1001 The output bytes from this mechanism are taken from the start of the final cipher block produced in the
 1002 MACing process.

1003 Constraints on key types and the length of input and output data are summarized in the following table:

1004 *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

1005 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1006 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for
 1007 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1008 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF
 1009 ciphers, these fields are not used.

1010 **2.7.14 General block cipher MAC**

1011 Cipher <NAME> has a MACing mechanism, “<NAME>-MAC”, denoted **CKM_<NAME>_MAC**. This
 1012 mechanism is a special case of the **CKM_<NAME>_MAC_GENERAL** mechanism described above. It
 1013 produces an output of size half as large as <NAME>’s blocksize.

1014 This mechanism has no parameters.

1015 Constraints on key types and the length of data are summarized in the following table:

1016 *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]

1017 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1018 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for
 1019 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1020 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF
 1021 ciphers, these fields are not used.

1022 2.8 SKIPJACK

1023 2.8.1 Definitions

1024 This section defines the key type “CKK_SKIPJACK” for type CK_KEY_TYPE as used in the
1025 CKA_KEY_TYPE attribute of key objects.

1026 Mechanisms:

- 1027 CKM_SKIPJACK_KEY_GEN
- 1028 CKM_SKIPJACK_ECB64
- 1029 CKM_SKIPJACK_CBC64
- 1030 CKM_SKIPJACK_OFB64
- 1031 CKM_SKIPJACK_CFB64
- 1032 CKM_SKIPJACK_CFB32
- 1033 CKM_SKIPJACK_CFB16
- 1034 CKM_SKIPJACK_CFB8
- 1035 CKM_SKIPJACK_WRAP
- 1036 CKM_SKIPJACK_PRIVATE_WRAP
- 1037 CKM_SKIPJACK_RELAYX

1038 2.8.2 SKIPJACK secret key objects

1039 SKIPJACK secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_SKIPJACK**) holds a
1040 single-length MEK or a TEK. The following table defines the SKIPJACK secret object attributes, in
1041 addition to the common attributes defined for this object class:

1042 *Table 31, SKIPJACK Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (12 bytes long)

1043 Refer to [PKCS #11-Base] table 10 for footnotes

1044

1045 SKIPJACK keys have 16 checksum bits, and these bits must be properly set. Attempting to create or
1046 unwrap a SKIPJACK key with incorrect checksum bits MUST return an error.

1047 It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK
1048 key with a specified value. Nonetheless, we provide templates for doing so.

1049 The following is a sample template for creating a SKIPJACK MEK secret key object:

```
1050 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1051 CK_KEY_TYPE keyType = CKK_SKIPJACK;  
1052 CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";  
1053 CK_BYTE value[12] = {...};  
1054 CK_BBOOL true = CK_TRUE;  
1055 CK_ATTRIBUTE template[] = {  
1056     {CKA_CLASS, &class, sizeof(class)},  
1057     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1058     {CKA_TOKEN, &>true, sizeof(true)},  
1059     {CKA_LABEL, label, sizeof(label)-1},  
1060     {CKA_ENCRYPT, &>true, sizeof(true)},  
1061     {CKA_VALUE, value, sizeof(value)}  
1062 };
```

1063 The following is a sample template for creating a SKIPJACK TEK secret key object:

```

1064 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1065 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1066 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1067 CK_BYTE value[12] = {...};
1068 CK_BBOOL true = CK_TRUE;
1069 CK_ATTRIBUTE template[] = {
1070     {CKA_CLASS, &class, sizeof(class)},
1071     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1072     {CKA_TOKEN, &>true, sizeof(true)},
1073     {CKA_LABEL, label, sizeof(label)-1},
1074     {CKA_ENCRYPT, &>true, sizeof(true)},
1075     {CKA_WRAP, &>true, sizeof(true)},
1076     {CKA_VALUE, value, sizeof(value)}
1077 };

```

1078 2.8.3 SKIPJACK Mechanism parameters

1079 2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; 1080 CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR

1081 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS** is a structure that provides the parameters to the
1082 **CKM_SKIPJACK_PRIVATE_WRAP** mechanism. It is defined as follows:

```

1083 typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
1084     CK_ULONG ulPasswordLen;
1085     CK_BYTE_PTR pPassword;
1086     CK_ULONG ulPublicDataLen;
1087     CK_BYTE_PTR pPublicData;
1088     CK_ULONG ulPandGLen;
1089     CK_ULONG ulQLen;
1090     CK_ULONG ulRandomLen;
1091     CK_BYTE_PTR pRandomA;
1092     CK_BYTE_PTR pPrimeP;
1093     CK_BYTE_PTR pBaseG;
1094     CK_BYTE_PTR pSubprimeQ;
1095 } CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

```

1096 The fields of the structure have the following meanings:

1097	<i>ulPasswordLen</i>	length of the password
1098	<i>pPassword</i>	pointer to the buffer which contains the user-supplied password
1099		
1100	<i>ulPublicDataLen</i>	other party's key exchange public key size
1101	<i>pPublicData</i>	pointer to other party's key exchange public key value
1102	<i>ulPandGLen</i>	length of prime and base values
1103	<i>ulQLen</i>	length of subprime value
1104	<i>ulRandomLen</i>	size of random Ra, in bytes
1105	<i>pPrimeP</i>	pointer to Prime, p, value
1106	<i>pBaseG</i>	pointer to Base, b, value

1107 *pSubprimeQ* pointer to Subprime, q, value

1108 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR** is a pointer to a
1109 **CK_PRIVATE_WRAP_PARAMS**.

1110 **2.8.3.2 CK_SKIPJACK_RELAYX_PARAMS;** 1111 **CK_SKIPJACK_RELAYX_PARAMS_PTR**

1112 **CK_SKIPJACK_RELAYX_PARAMS** is a structure that provides the parameters to the
1113 **CKM_SKIPJACK_RELAYX** mechanism. It is defined as follows:

```
1114 typedef struct CK_SKIPJACK_RELAYX_PARAMS {  
1115     CK_ULONG ulOldWrappedXLen;  
1116     CK_BYTE_PTR pOldWrappedX;  
1117     CK_ULONG ulOldPasswordLen;  
1118     CK_BYTE_PTR pOldPassword;  
1119     CK_ULONG ulOldPublicDataLen;  
1120     CK_BYTE_PTR pOldPublicData;  
1121     CK_ULONG ulOldRandomLen;  
1122     CK_BYTE_PTR pOldRandomA;  
1123     CK_ULONG ulNewPasswordLen;  
1124     CK_BYTE_PTR pNewPassword;  
1125     CK_ULONG ulNewPublicDataLen;  
1126     CK_BYTE_PTR pNewPublicData;  
1127     CK_ULONG ulNewRandomLen;  
1128     CK_BYTE_PTR pNewRandomA;  
1129 } CK_SKIPJACK_RELAYX_PARAMS;
```

1130 The fields of the structure have the following meanings:

1131 *ulOldWrappedLen* length of old wrapped key in bytes

1132 *pOldWrappedX* pointer to old wrapper key

1133 *ulOldPasswordLen* length of the old password

1134 *pOldPassword* pointer to the buffer which contains the old user-supplied
1135 password

1136 *ulOldPublicDataLen* old key exchange public key size

1137 *pOldPublicData* pointer to old key exchange public key value

1138 *ulOldRandomLen* size of old random Ra in bytes

1139 *pOldRandomA* pointer to old Ra data

1140 *ulNewPasswordLen* length of the new password

1141 *pNewPassword* pointer to the buffer which contains the new user-
1142 supplied password

1143 *ulNewPublicDataLen* new key exchange public key size

1144 *pNewPublicData* pointer to new key exchange public key value

1145 *ulNewRandomLen* size of new random Ra in bytes

1146 *pNewRandomA* pointer to new Ra data

1147 **CK_SKIPJACK_RELAYX_PARAMS_PTR** is a pointer to a **CK_SKIPJACK_RELAYX_PARAMS**.

1148 2.8.4 SKIPJACK key generation

1149 The SKIPJACK key generation mechanism, denoted **CKM_SKIPJACK_KEY_GEN**, is a key generation
1150 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1151 It does not have a parameter.

1152 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
1153 key.

1154 2.8.5 SKIPJACK-ECB64

1155 SKIPJACK-ECB64, denoted **CKM_SKIPJACK_ECB64**, is a mechanism for single- and multiple-part
1156 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB
1157 185.

1158 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1159 value generated by the token – in other words, the application cant specify a particular IV when
1160 encrypting. It MAY, of course, specify a particular IV when decrypting.

1161 Constraints on key types and the length of data are summarized in the following table:

1162 *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

1163 2.8.6 SKIPJACK-CBC64

1164 SKIPJACK-CBC64, denoted **CKM_SKIPJACK_CBC64**, is a mechanism for single- and multiple-part
1165 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1166 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1167 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1168 encrypting. It MAY, of course, specify a particular IV when decrypting.

1169 Constraints on key types and the length of data are summarized in the following table:

1170 *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

1171 2.8.7 SKIPJACK-OFB64

1172 SKIPJACK-OFB64, denoted **CKM_SKIPJACK_OFB64**, is a mechanism for single- and multiple-part
1173 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1174 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1175 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1176 encrypting. It MAY, of course, specify a particular IV when decrypting.

1177 Constraints on key types and the length of data are summarized in the following table:

1178 *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

1179 **2.8.8 SKIPJACK-CFB64**

1180 SKIPJACK-CFB64, denoted **CKM_SKIPJACK_CFB64**, is a mechanism for single- and multiple-part
1181 encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.

1182 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1183 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1184 encrypting. It MAY, of course, specify a particular IV when decrypting.

1185 Constraints on key types and the length of data are summarized in the following table:

1186 *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

1187 **2.8.9 SKIPJACK-CFB32**

1188 SKIPJACK-CFB32, denoted **CKM_SKIPJACK_CFB32**, is a mechanism for single- and multiple-part
1189 encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.

1190 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1191 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1192 encrypting. It MAY, of course, specify a particular IV when decrypting.

1193 Constraints on key types and the length of data are summarized in the following table:

1194 *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

1195 **2.8.10 SKIPJACK-CFB16**

1196 SKIPJACK-CFB16, denoted **CKM_SKIPJACK_CFB16**, is a mechanism for single- and multiple-part
1197 encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.

1198 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1199 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1200 encrypting. It MAY, of course, specify a particular IV when decrypting.

1201 Constraints on key types and the length of data are summarized in the following table:

1202 *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
-----------	----------	---------------	----------------------	---------------

1203 2.8.11 SKIPJACK-CFB8

1204 SKIPJACK-CFB8, denoted **CKM_SKIPJACK_CFB8**, is a mechanism for single- and multiple-part
1205 encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.

1206 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1207 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1208 encrypting. It MAY, of course, specify a particular IV when decrypting.

1209 Constraints on key types and the length of data are summarized in the following table:

1210 *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

1211 2.8.12 SKIPJACK-WRAP

1212 The SKIPJACK-WRAP mechanism, denoted **CKM_SKIPJACK_WRAP**, is used to wrap and unwrap a
1213 secret key (MEK). It MAY wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.

1214 It does not have a parameter.

1215 2.8.13 SKIPJACK-PRIVATE-WRAP

1216 The SKIPJACK-PRIVATE-WRAP mechanism, denoted **CKM_SKIPJACK_PRIVATE_WRAP**, is used to
1217 wrap and unwrap a private key. It MAY wrap KEA and DSA private keys.

1218 It has a parameter, a **CK_SKIPJACK_PRIVATE_WRAP_PARAMS** structure.

1219 2.8.14 SKIPJACK-RELAYX

1220 The SKIPJACK-RELAYX mechanism, denoted **CKM_SKIPJACK_RELAYX**, is used with the **C_WrapKey**
1221 function to “change the wrapping” on a private key which was wrapped with the SKIPJACK-PRIVATE-
1222 WRAP mechanism (See Section 2.8.13).

1223 It has a parameter, a **CK_SKIPJACK_RELAYX_PARAMS** structure.

1224 Although the SKIPJACK-RELAYX mechanism is used with **C_WrapKey**, it differs from other key-
1225 wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to
1226 **C_WrapKey**; however for the SKIPJACK_RELAYX mechanism, the [always invalid] value 0 should be
1227 passed as the key handle for **C_WrapKey**, and the already-wrapped key should be passed in as part of
1228 the **CK_SKIPJACK_RELAYX_PARAMS** structure.

1229 2.9 BATON

1230 2.9.1 Definitions

1231 This section defines the key type “CKK_BATON” for type CK_KEY_TYPE as used in the
1232 CKA_KEY_TYPE attribute of key objects.

1233 Mechanisms:

1234 CKM_BATON_KEY_GEN

1235 CKM_BATON_ECB128

1236 CKM_BATON_ECB96

1237 CKM_BATON_CBC128
1238 CKM_BATON_COUNTER
1239 CKM_BATON_SHUFFLE
1240 CKM_BATON_WRAP

1241 2.9.2 BATON secret key objects

1242 BATON secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_BATON**) hold single-length
1243 BATON keys. The following table defines the BATON secret key object attributes, in addition to the
1244 common attributes defined for this object class:

1245 *Table 39, BATON Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (40 bytes long)

1246 Refer to [PKCS #11-Base] table 10 for footnotes

1247

1248 BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or
1249 unwrap a BATON key with incorrect checksum bits MUST return an error.

1250 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key
1251 with a specified value. Nonetheless, we provide templates for doing so.

1252 The following is a sample template for creating a BATON MEK secret key object:

```
1253 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1254 CK_KEY_TYPE keyType = CKK_BATON;  
1255 CK_UTF8CHAR label[] = "A BATON MEK secret key object";  
1256 CK_BYTE value[40] = {...};  
1257 CK_BBOOL true = CK_TRUE;  
1258 CK_ATTRIBUTE template[] = {  
1259     {CKA_CLASS, &class, sizeof(class)},  
1260     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1261     {CKA_TOKEN, &>true, sizeof(true)},  
1262     {CKA_LABEL, label, sizeof(label)-1},  
1263     {CKA_ENCRYPT, &>true, sizeof(true)},  
1264     {CKA_VALUE, value, sizeof(value)}  
1265 };
```

1266 The following is a sample template for creating a BATON TEK secret key object:

```
1267 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1268 CK_KEY_TYPE keyType = CKK_BATON;  
1269 CK_UTF8CHAR label[] = "A BATON TEK secret key object";  
1270 CK_BYTE value[40] = {...};  
1271 CK_BBOOL true = CK_TRUE;  
1272 CK_ATTRIBUTE template[] = {  
1273     {CKA_CLASS, &class, sizeof(class)},  
1274     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1275     {CKA_TOKEN, &>true, sizeof(true)},  
1276     {CKA_LABEL, label, sizeof(label)-1},  
1277     {CKA_ENCRYPT, &>true, sizeof(true)},  
1278     {CKA_WRAP, &>true, sizeof(true)},  
1279     {CKA_VALUE, value, sizeof(value)}  
1280 };
```

1281 2.9.3 BATON key generation

1282 The BATON key generation mechanism, denoted **CKM_BATON_KEY_GEN**, is a key generation
1283 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

1284 It does not have a parameter.
 1285 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
 1286 key.

1287 **2.9.4 BATON-ECB128**

1288 BATON-ECB128, denoted **CKM_BATON_ECB128**, is a mechanism for single- and multiple-part
 1289 encryption and decryption with BATON in 128-bit electronic codebook mode.
 1290 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1291 value generated by the token – in other words, the application MAY NOT specify a particular IV when
 1292 encrypting. It MAY, of course, specify a particular IV when decrypting.
 1293 Constraints on key types and the length of data are summarized in the following table:

1294 *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

1295 **2.9.5 BATON-ECB96**

1296 BATON-ECB96, denoted **CKM_BATON_ECB96**, is a mechanism for single- and multiple-part encryption
 1297 and decryption with BATON in 96-bit electronic codebook mode.
 1298 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1299 value generated by the token – in other words, the application MAY NOT specify a particular IV when
 1300 encrypting. It MAY, of course, specify a particular IV when decrypting.

1301 Constraints on key types and the length of data are summarized in the following table:

1302 *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

1303 **2.9.6 BATON-CBC128**

1304 BATON-CBC128, denoted **CKM_BATON_CBC128**, is a mechanism for single- and multiple-part
 1305 encryption and decryption with BATON in 128-bit cipher-block chaining mode.
 1306 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1307 value generated by the token – in other words, the application MAY NOT specify a particular IV when
 1308 encrypting. It MAY, of course, specify a particular IV when decrypting.

1309 Constraints on key types and the length of data are summarized in the following table:

1310 *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

1311 2.9.7 BATON-COUNTER

1312 BATON-COUNTER, denoted **CKM_BATON_COUNTER**, is a mechanism for single- and multiple-part
1313 encryption and decryption with BATON in counter mode.

1314 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1315 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1316 encrypting. It MAY, of course, specify a particular IV when decrypting.

1317 Constraints on key types and the length of data are summarized in the following table:

1318 *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

1319 2.9.8 BATON-SHUFFLE

1320 BATON-SHUFFLE, denoted **CKM_BATON_SHUFFLE**, is a mechanism for single- and multiple-part
1321 encryption and decryption with BATON in shuffle mode.

1322 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1323 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1324 encrypting. It MAY, of course, specify a particular IV when decrypting.

1325 Constraints on key types and the length of data are summarized in the following table:

1326 *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

1327 2.9.9 BATON WRAP

1328 The BATON wrap and unwrap mechanism, denoted **CKM_BATON_WRAP**, is a function used to wrap
1329 and unwrap a secret key (MEK). It MAY wrap and unwrap SKIPJACK, BATON and JUNIPER keys.

1330 It has no parameters.

1331 When used to unwrap a key, this mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and
1332 **CKA_VALUE** attributes to it.

1333 2.10 JUNIPER

1334 2.10.1 Definitions

1335 This section defines the key type “CKK_JUNIPER” for type CK_KEY_TYPE as used in the
1336 CKA_KEY_TYPE attribute of key objects.

1337 Mechanisms:

1338 CKM_JUNIPER_KEY_GEN

1339 CKM_JUNIPER_ECB128

1340 CKM_JUNIPER_CBC128

1341 CKM_JUNIPER_COUNTER

1342 CKM_JUNIPER_SHUFFLE

1343 CKM_JUNIPER_WRAP

1344 2.10.2 JUNIPER secret key objects

1345 JUNIPER secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_JUNIPER**) hold single-
1346 length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to
1347 the common attributes defined for this object class:

1348 *Table 45, JUNIPER Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE ^{1,4,6,7}	Byte array	Key value (40 bytes long)

1349 Refer to [PKCS #11-Base] table 10 for footnotes

1350

1351 JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or
1352 unwrap a BATON key with incorrect checksum bits MUST return an error.

1353 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key
1354 with a specified value. Nonetheless, we provide templates for doing so.

1355 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1356 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1357 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1358 CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";  
1359 CK_BYTE value[40] = {...};  
1360 CK_BBOOL true = CK_TRUE;  
1361 CK_ATTRIBUTE template[] = {  
1362     {CKA_CLASS, &class, sizeof(class)},  
1363     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1364     {CKA_TOKEN, &>true, sizeof(true)},  
1365     {CKA_LABEL, label, sizeof(label)-1},  
1366     {CKA_ENCRYPT, &>true, sizeof(true)},  
1367     {CKA_VALUE, value, sizeof(value)}  
1368 };
```

1369 The following is a sample template for creating a JUNIPER TEK secret key object:

```
1370 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1371 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1372 CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";  
1373 CK_BYTE value[40] = {...};  
1374 CK_BBOOL true = CK_TRUE;  
1375 CK_ATTRIBUTE template[] = {  
1376     {CKA_CLASS, &class, sizeof(class)},  
1377     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1378     {CKA_TOKEN, &>true, sizeof(true)},  
1379     {CKA_LABEL, label, sizeof(label)-1},  
1380     {CKA_ENCRYPT, &>true, sizeof(true)},  
1381     {CKA_WRAP, &>true, sizeof(true)},  
1382     {CKA_VALUE, value, sizeof(value)}  
1383 };
```

1384 2.10.3 JUNIPER key generation

1385 The JUNIPER key generation mechanism, denoted **CKM_JUNIPER_KEY_GEN**, is a key generation
1386 mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).

1387 It does not have a parameter.

1388 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
1389 key.

1390 2.10.4 JUNIPER-ECB128

1391 JUNIPER-ECB128, denoted **CKM_JUNIPER_ECB128**, is a mechanism for single- and multiple-part
1392 encryption and decryption with JUNIPER in 128-bit electronic codebook mode.

1393 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1394 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1395 encrypting. It MAY, of course, specify a particular IV when decrypting.

1396 Constraints on key types and the length of data are summarized in the following table. For encryption
1397 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1398 *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

1399 2.10.5 JUNIPER-CBC128

1400 JUNIPER-CBC128, denoted **CKM_JUNIPER_CBC128**, is a mechanism for single- and multiple-part
1401 encryption and decryption with JUNIPER in 128-bit cipher block chaining mode.

1402 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1403 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1404 encrypting. It MAY, of course, specify a particular IV when decrypting.

1405 Constraints on key types and the length of data are summarized in the following table. For encryption
1406 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1407 *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

1408 2.10.6 JUNIPER-COUNTER

1409 JUNIPER-COUNTER, denoted **CKM_JUNIPER_COUNTER**, is a mechanism for single- and multiple-
1410 part encryption and decryption with JUNIPER in counter mode.

1411 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1412 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1413 encrypting. It MAY, of course, specify a particular IV when decrypting.

1414 Constraints on key types and the length of data are summarized in the following table. For encryption
1415 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1416 *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

1417 2.10.7 JUNIPER-SHUFFLE

1418 JUNIPER-SHUFFLE, denoted **CKM_JUNIPER_SHUFFLE**, is a mechanism for single- and multiple-part
1419 encryption and decryption with JUNIPER in shuffle mode.

1420 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1421 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1422 encrypting. It MAY, of course, specify a particular IV when decrypting.

1423 Constraints on key types and the length of data are summarized in the following table. For encryption
1424 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1425 *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

1426 2.10.8 JUNIPER WRAP

1427 The JUNIPER wrap and unwrap mechanism, denoted **CKM_JUNIPER_WRAP**, is a function used to wrap
1428 and unwrap an MEK. It MAY wrap or unwrap SKIPJACK, BATON and JUNIPER keys.

1429 It has no parameters.

1430 When used to unwrap a key, this mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and
1431 **CKA_VALUE** attributes to it.

1432 2.11 MD2

1433 2.11.1 Definitions

1434 Mechanisms:

1435 CKM_MD2

1436 CKM_MD2_HMAC

1437 CKM_MD2_HMAC_GENERAL

1438 CKM_MD2_KEY_DERIVATION

1439 2.11.2 MD2 digest

1440 The MD2 mechanism, denoted **CKM_MD2**, is a mechanism for message digesting, following the MD2
1441 message-digest algorithm defined in RFC 6149.

1442 It does not have a parameter.

1443 Constraints on the length of data are summarized in the following table:

1444 *Table 50, MD2: Data Length*

Function	Data length	Digest Length
C_Digest	Any	16

1445 2.11.3 General-length MD2-HMAC

1446 The general-length MD2-HMAC mechanism, denoted **CKM_MD2_HMAC_GENERAL**, is a mechanism for
1447 signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it
1448 uses are generic secret keys.

1449 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1450 output. This length should be in the range 0-16 (the output size of MD2 is 16 bytes). Signatures (MACs)
1451 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1452 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

1453 2.11.4 MD2-HMAC

1454 The MD2-HMAC mechanism, denoted **CKM_MD2_HMAC**, is a special case of the general-length MD2-
1455 HMAC mechanism in Section 2.11.3.

1456 It has no parameter, and produces an output of length 16.

1457 2.11.5 MD2 key derivation

1458 MD2 key derivation, denoted **CKM_MD2_KEY_DERIVATION**, is a mechanism which provides the
1459 capability of deriving a secret key by digesting the value of another secret key with MD2.

1460 The value of the base key is digested once, and the result is used to make the value of the derived secret
1461 key.

- 1462 • If no length or key type is provided in the template, then the key produced by this mechanism **MUST**
1463 be a generic secret key. Its length **MUST** be 16 bytes (the output size of MD2)..
- 1464 • If no key type is provided in the template, but a length is, then the key produced by this mechanism
1465 **MUST** be a generic secret key of the specified length.
- 1466 • If no length was provided in the template, but a key type is, then that key type must have a well-
1467 defined length. If it does, then the key produced by this mechanism **MUST** be of the type specified in
1468 the template. If it doesn't, an error **MUST** be returned.
- 1469 • If both a key type and a length are provided in the template, the length must be compatible with that
1470 key type. The key produced by this mechanism **MUST** be of the specified type and length.

1471 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key **MUST** be set
1472 properly.

1473 If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.

1474 This mechanism has the following rules about key sensitivity and extractability:

- 1475 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key **MAY**
1476 both be specified to be either **CK_TRUE** or **CK_FALSE**. If omitted, these attributes each take on
1477 some default value.
- 1478 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to **CK_FALSE**, then the derived key
1479 **MUST** as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to **CK_TRUE**, then
1480 the derived key has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value as its
1481 **CKA_SENSITIVE** attribute.
- 1482 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to **CK_FALSE**, then the
1483 derived key **MUST**, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to
1484 **CK_TRUE**, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the *opposite*
1485 value from its **CKA_EXTRACTABLE** attribute.

1486 2.12 MD5

1487 2.12.1 Definitions

1488 Mechanisms:

1489 CKM_MD5

1490 CKM_MD5_HMAC

1491 CKM_MD5_HMAC_GENERAL
1492 CKM_MD5_KEY_DERIVATION

1493 2.12.2 MD5 Digest

1494 The MD5 mechanism, denoted **CKM_MD5**, is a mechanism for message digesting, following the MD5
1495 message-digest algorithm defined in RFC 1321.

1496 It does not have a parameter.

1497 Constraints on the length of input and output data are summarized in the following table. For single-part
1498 digesting, the data and the digest MAY begin at the same location in memory.

1499 *Table 52, MD5: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1500 2.12.3 General-length MD5-HMAC

1501 The general-length MD5-HMAC mechanism, denoted **CKM_MD5_HMAC_GENERAL**, is a mechanism for
1502 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it
1503 uses are generic secret keys.

1504 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1505 output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)
1506 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1507 *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

1508 2.12.4 MD5-HMAC

1509 The MD5-HMAC mechanism, denoted **CKM_MD5_HMAC**, is a special case of the general-length MD5-
1510 HMAC mechanism in Section 2.12.3.

1511 It has no parameter, and produces an output of length 16.

1512 2.12.5 MD5 key derivation

1513 MD5 key derivation denoted **CKM_MD5_KEY_DERIVATION**, is a mechanism which provides the
1514 capability of deriving a secret key by digesting the value of another secret key with MD5.

1515 The value of the base key is digested once, and the result is used to make the value of derived secret
1516 key.

- 1517 • If no length or key type is provided in the template, then the key produced by this mechanism MUST
1518 be a generic secret key. Its length MUST be 16 bytes (the output size of MD5).
- 1519 • If no key type is provided in the template, but a length is, then the key produced by this mechanism
1520 MUST be a generic secret key of the specified length.
- 1521 • If no length was provided in the template, but a key type is, then that key type must have a well-
1522 defined length. If it does, then the key produced by this mechanism MUST be of the type specified in
1523 the template. If it doesn't, an error MUST be returned.
- 1524 • If both a key type and a length are provided in the template, the length must be compatible with that
1525 key type. The key produced by this mechanism MUST be of the specified type and length.

1526 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set
 1527 properly.

1528 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.

1529 This mechanism has the following rules about key sensitivity and extractability.

- 1530 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key MAY
 1531 both be specified to either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some
 1532 default value.
- 1533 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, then the derived key
 1534 MUST as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_TRUE, then
 1535 the derived key has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value as its
 1536 **CKA_SENSITIVE** attribute.
- 1537 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE, then the
 1538 derived key MUST, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to
 1539 CK_TRUE, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the *opposite*
 1540 value from its **CKA_EXTRACTABLE** attribute.

1541 2.13 FASTHASH

1542 2.13.1 Definitions

1543 Mechanisms:
 1544 CKM_FASTHASH

1545 2.13.2 FASTHASH digest

1546 The FASTHASH mechanism, denoted **CKM_FASTHASH**, is a mechanism for message digesting,
 1547 following the U.S. government's algorithm.

1548 It does not have a parameter.

1549 Constraints on the length of input and output data are summarized in the following table:

1550 *Table 54, FASTHASH: Data Length*

Function	Input length	Digest length
C_Digest	Any	40

1551 2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)

1552 2.14.1 Definitions

1553 The mechanisms in this section are for generating keys and IVs for performing password-based
 1554 encryption. The method used to generate keys and IVs is specified in PKCS #5.

1555 Mechanisms:

- 1556 CKM_PBE_MD2_DES_CBC
- 1557 CKM_PBE_MD5_DES_CBC
- 1558 CKM_PBE_MD5_CAST_CBC
- 1559 CKM_PBE_MD5_CAST3_CBC
- 1560 CKM_PBE_MD5_CAST5_CBC
- 1561 CKM_PBE_MD5_CAST128_CBC
- 1562 CKM_PBE_SHA1_CAST5_CBC
- 1563 CKM_PBE_SHA1_CAST128_CBC

1564 CKM_PBE_SHA1_RC4_128
1565 CKM_PBE_SHA1_RC4_40
1566 CKM_PBE_SHA1_RC2_128_CBC
1567 CKM_PBE_SHA1_RC2_40_CBC

1568 2.14.2 Password-based encryption/authentication mechanism parameters

1569 2.14.2.1 CK_PBE_PARAMS; CK_PBE_PARAMS_PTR

1570 **CK_PBE_PARAMS** is a structure which provides all of the necessary information required by the
1571 CKM_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation
1572 mechanisms) and the CKM_PBA_SHA1_WITH_SHA1_HMAC mechanism. It is defined as follows:

```
1573 typedef struct CK_PBE_PARAMS {  
1574     CK_BYTE_PTR pInitVector;  
1575     CK_UTF8CHAR_PTR pPassword;  
1576     CK_ULONG ulPasswordLen;  
1577     CK_BYTE_PTR pSalt;  
1578     CK_ULONG ulSaltLen;  
1579     CK_ULONG ulIteration;  
1580 } CK_PBE_PARAMS;
```

1581 The fields of the structure have the following meanings:

1582	<i>pInitVector</i>	pointer to the location that receives the 8-byte initialization vector (IV), if an IV is required
1583		
1584	<i>pPassword</i>	points to the password to be used in the PBE key generation
1585		
1586	<i>ulPasswordLen</i>	length in bytes of the password information
1587	<i>pSalt</i>	points to the salt to be used in the PBE key generation
1588	<i>ulSaltLen</i>	length in bytes of the salt information
1589	<i>ulliteration</i>	number of iterations required for the generation

1590 **CK_PBE_PARAMS_PTR** is a pointer to a **CK_PBE_PARAMS**.

1591 2.14.3 MD2-PBE for DES-CBC

1592 MD2-PBE for DES-CBC, denoted **CKM_PBE_MD2_DES_CBC**, is a mechanism used for generating a
1593 DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an
1594 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1595 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1596 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1597 generated by the mechanism.

1598 2.14.4 MD5-PBE for DES-CBC

1599 MD5-PBE for DES-CBC, denoted **CKM_PBE_MD5_DES_CBC**, is a mechanism used for generating a
1600 DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
1601 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1602 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1603 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1604 generated by the mechanism.

1605 **2.14.5 MD5-PBE for CAST-CBC**

1606 MD5-PBE for CAST-CBC, denoted **CKM_PBE_MD5_CAST_CBC**, is a mechanism used for generating a
1607 CAST secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
1608 iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1609 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1610 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1611 generated by the mechanism

1612 The length of the CAST key generated by this mechanism MAY be specified in the supplied template; if it
1613 is not present in the template, it defaults to 8 bytes.

1614 **2.14.6 MD5-PBE for CAST3-CBC**

1615 MD5-PBE for CAST3-CBC, denoted **CKM_PBE_MD5_CAST3_CBC**, is a mechanism used for generating
1616 a CAST3 secret key and an IV from a password and a salt value by using the MD5 digest algorithm and
1617 an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1618 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1619 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1620 generated by the mechanism

1621 The length of the CAST3 key generated by this mechanism MAY be specified in the supplied template; if
1622 it is not present in the template, it defaults to 8 bytes.

1623 **2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC)**

1624 MD5-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM_PBE_MD5_CAST128_CBC** or
1625 **CKM_PBE_MD5_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
1626 and an IV from a password and a salt value by using the MD5 digest algorithm and an iteration count.
1627 This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1628 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1629 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1630 generated by the mechanism

1631 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied
1632 template; if it is not present in the template, it defaults to 8 bytes.

1633 **2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)**

1634 SHA-1-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM_PBE_SHA1_CAST128_CBC** or
1635 **CKM_PBE_SHA1_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
1636 and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This
1637 functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1638 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1639 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1640 generated by the mechanism

1641 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied
1642 template; if it is not present in the template, it defaults to 8 bytes

1643 **2.15 PKCS #12 password-based encryption/authentication**
1644 **mechanisms**

1645 **2.15.1 Definitions**

1646 The mechanisms in this section are for generating keys and IVs for performing password-based
1647 encryption or authentication. The method used to generate keys and IVs is based on a method that was
1648 specified in PKCS #12.

1649 We specify here a general method for producing various types of pseudo-random bits from a password,
1650 p ; a string of salt bits, s ; and an iteration count, c . The “type” of pseudo-random bits to be produced is
1651 identified by an identification byte, ID , described at the end of this section.

1652 Let H be a hash function built around a compression function $f: \mathbb{Z}_2^u \times \mathbb{Z}_2^v \rightarrow \mathbb{Z}_2^u$ (that is, H has a chaining
1653 variable and output of length u bits, and the message input to the compression function of H is v bits). For
1654 MD2 and MD5, $u=128$ and $v=512$; for SHA-1, $u=160$ and $v=512$.

1655 We assume here that u and v are both multiples of 8, as are the lengths in bits of the password and salt
1656 strings and the number n of pseudo-random bits required. In addition, u and v are of course nonzero.

- 1657 1. Construct a string, D (the “diversifier”), by concatenating $v/8$ copies of ID .
- 1658 2. Concatenate copies of the salt together to create a string S of length $v \lceil s/v \rceil$ bits (the final copy of
1659 the salt MAY be truncated to create S). Note that if the salt is the empty string, then so is S .
- 1660 3. Concatenate copies of the password together to create a string P of length $v \lceil p/v \rceil$ bits (the final
1661 copy of the password MAY be truncated to create P). Note that if the password is the empty
1662 string, then so is P .
- 1663 4. Set $I=S||P$ to be the concatenation of S and P .
- 1664 5. Set $j=\lceil n/u \rceil$.
- 1665 6. For $i=1, 2, \dots, j$, do the following:
 - 1666 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
1667 of that hash; etc.; continue in this fashion until a total of c hashes have been computed,
1668 each on the result of the previous hash.
 - 1669 b. Concatenate copies of A_i to create a string B of length v bits (the final copy of A_i MAY be
1670 truncated to create B).
 - 1671 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
1672 by setting $I_j=(I_j+B+1) \bmod 2^v$ for each j . To perform this addition, treat each v -bit block as
1673 a binary number represented most-significant bit first.
- 1674 7. Concatenate A_1, A_2, \dots, A_j together to form a pseudo-random bit string, A .
- 1675 8. Use the first n bits of A as the output of this entire process

1676 When the password-based encryption mechanisms presented in this section are used to generate a key
1677 and IV (if needed) from a password, salt, and an iteration count, the above algorithm is used. To
1678 generate a key, the identifier byte ID is set to the value 1; to generate an IV, the identifier byte ID is set to
1679 the value 2.

1680 When the password-based authentication mechanism presented in this section is used to generate a key
1681 from a password, salt and an iteration count, the above algorithm is used. The identifier ID is set to the
1682 value 3.

1683 **2.15.2 SHA-1-PBE for 128-bit RC4**

1684 SHA-1-PBE for 128-bit RC4, denoted **CKM_PBE_SHA1_RC4_128**, is a mechanism used for generating
1685 a 128-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
1686 iteration count. The method used to generate the key is described above.

1687 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1688 key generation process. The parameter also has a field to hold the location of an application-supplied
1689 buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not
1690 require an IV.
1691 The key produced by this mechanism will typically be used for performing password-based encryption.

1692 **2.15.3 SHA-1_PBE for 40-bit RC4**

1693 SHA-1-PBE for 40-bit RC4, denoted **CKM_PBE_SHA1_RC4_40**, is a mechanism used for generating a
1694 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
1695 iteration count. The method used to generate the key is described above.

1696 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1697 key generation process. The parameter also has a field to hold the location of an application-supplied
1698 buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not
1699 require an IV.

1700 The key produced by this mechanism will typically be used for performing password-based encryption.

1701 **2.15.4 SHA-1_PBE for 128-bit RC2-CBC**

1702 SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_128_CBC**, is a mechanism used for
1703 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest
1704 algorithm and an iteration count. The method used to generate the key and IV is described above.

1705 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1706 key generation process and the location of an application-supplied buffer which receives the 8-byte IV
1707 generated by the mechanism.

1708 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
1709 of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object
1710 Identifier `pbeWithSHA1And128BitRC2-CBC`.

1711 The key and IV produced by this mechanism will typically be used for performing password-based
1712 encryption.

1713 **2.15.5 SHA-1_PBE for 40-bit RC2-CBC**

1714 SHA-1-PBE for 40-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_40_CBC**, is a mechanism used for
1715 generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm
1716 and an iteration count. The method used to generate the key and IV is described above.

1717 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1718 key generation process and the location of an application-supplied buffer which receives the 8-byte IV
1719 generated by the mechanism.

1720 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
1721 of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object
1722 Identifier `pbeWithSHA1And40BitRC2-CBC`.

1723 The key and IV produced by this mechanism will typically be used for performing password-based
1724 encryption

1725 **2.16 RIPE-MD**

1726 **2.16.1 Definitions**

1727 Mechanisms:

1728 **CKM_RIPEMD128**

1729 **CKM_RIPEMD128_HMAC**

1730 CKM_RIPEMD128_HMAC_GENERAL
 1731 CKM_RIPEMD160
 1732 CKM_RIPEMD160_HMAC
 1733 CKM_RIPEMD160_HMAC_GENERAL

1734 2.16.2 RIPE-MD 128 Digest

1735 The RIPE-MD 128 mechanism, denoted **CKM_RIPEMD128**, is a mechanism for message
 1736 digesting, following the RIPE-MD 128 message-digest algorithm.

1737 It does not have a parameter.

1738 Constraints on the length of data are summarized in the following table:

1739 *Table 55, RIPE-MD 128: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1740

1741 2.16.3 General-length RIPE-MD 128-HMAC

1742 The general-length RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC_GENERAL**, is
 1743 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 128
 1744 hash function. The keys it uses are generic secret keys.

1745 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
 1746 output. This length should be in the range 0-16 (the output size of RIPE-MD 128 is 16 bytes). Signatures
 1747 (MACs) produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1748 *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

1749 2.16.4 RIPE-MD 128-HMAC

1750 The RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC**, is a special case of the
 1751 general-length RIPE-MD 128-HMAC mechanism in Section 2.16.3.

1752 It has no parameter, and produces an output of length 16.

1753 2.16.5 RIPE-MD 160

1754 The RIPE-MD 160 mechanism, denoted **CKM_RIPEMD160**, is a mechanism for message digesting,
 1755 following the RIPE-MD 160 message-digest defined in ISO-10118.

1756 It does not have a parameter.

1757 Constraints on the length of data are summarized in the following table:

1758 *Table 57, RIPE-MD 160: Data Length*

Function	Data length	Digest length
C_Digest	Any	20

1759 **2.16.6 General-length RIPE-MD 160-HMAC**

1760 The general-length RIPE-MD 160-HMAC mechanism, denoted **CKM_RIPEMD160_HMAC_GENERAL**, is
1761 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160
1762 hash function. The keys it uses are generic secret keys.

1763 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1764 output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures
1765 (MACs) produced by this mechanism **MUST** be taken from the start of the full 20-byte HMAC output.

1766 *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

1767 **2.16.7 RIPE-MD 160-HMAC**

1768 The RIPE-MD 160-HMAC mechanism, denoted **CKM_RIPEMD160_HMAC**, is a special case of the
1769 general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.

1770 It has no parameter, and produces an output of length 20.

1771 **2.17 SET**

1772 **2.17.1 Definitions**

1773 Mechanisms:

1774 **CKM_KEY_WRAP_SET_OAEP**

1775 **2.17.2 SET mechanism parameters**

1776 **2.17.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS;**
1777 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR**

1778 **CK_KEY_WRAP_SET_OAEP_PARAMS** is a structure that provides the parameters to the
1779 **CKM_KEY_WRAP_SET_OAEP** mechanism. It is defined as follows:

```

1780 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {
1781     CK_BYTE bBC;
1782     CK_BYTE_PTR pX;
1783     CK_ULONG ulXLen;
1784 } CK_KEY_WRAP_SET_OAEP_PARAMS;

```

1785 The fields of the structure have the following meanings:

- 1786 *bBC* block contents byte
- 1787 *pX* concatenation of hash of plaintext data (if present) and
1788 extra data (if present)
- 1789 *ulXLen* length in bytes of concatenation of hash of plaintext data
1790 (if present) and extra data (if present). 0 if neither is
1791 present.

1792 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR** is a pointer to a
1793 **CK_KEY_WRAP_SET_OAEP_PARAMS**.

1794 2.17.3 OAEP key wrapping for SET

1795 The OAEP key wrapping for SET mechanism, denoted **CKM_KEY_WRAP_SET_OAEP**, is a mechanism
1796 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some
1797 extra data MAY be wrapped together with the DES key. This mechanism is defined in the SET protocol
1798 specifications.

1799 It takes a parameter, a **CK_KEY_WRAP_SET_OAEP_PARAMS** structure. This structure holds the
1800 "Block Contents" byte of the data and the concatenation of the hash of plaintext data (if present) and the
1801 extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by
1802 the *ulXLen* field having the value 0.

1803 When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present)
1804 and the extra data (if present) is returned following the convention described [PKCS #11-Curr],
1805 **Miscellaneous simple key derivation mechanisms**. Note that if the inputs to **C_UnwrapKey** are such
1806 that the extra data is not returned (e.g. the buffer supplied in the
1807 **CK_KEY_WRAP_SET_OAEP_PARAMS** structure is **NULL_PTR**), then the unwrapped key object MUST
1808 NOT be created, either.

1809 Be aware that when this mechanism is used to unwrap a key, the *bBC* and *pX* fields of the parameter
1810 supplied to the mechanism MAY be modified.

1811 If an application uses **C_UnwrapKey** with **CKM_KEY_WRAP_SET_OAEP**, it may be preferable for it
1812 simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data
1813 (this concatenation MUST NOT be larger than 128 bytes), rather than calling **C_UnwrapKey** twice. Each
1814 call of **C_UnwrapKey** with **CKM_KEY_WRAP_SET_OAEP** requires an RSA decryption operation to be
1815 performed, and this computational overhead MAY be avoided by this means.

1816 2.18 LYNKS

1817 2.18.1 Definitions

1818 Mechanisms:

1819 **CKM_KEY_WRAP_LYNKS**

1820 2.18.2 LYNKS key wrapping

1821 The LYNKS key wrapping mechanism, denoted **CKM_KEY_WRAP_LYNKS**, is a mechanism for
1822 wrapping and unwrapping secret keys with DES keys. It MAY wrap any 8-byte secret key, and it produces
1823 a 10-byte wrapped key, containing a cryptographic checksum.

1824 It does not have a parameter.

1825 To wrap an 8-byte secret key *K* with a DES key *W*, this mechanism performs the following steps:

- 1826 1. Initialize two 16-bit integers, *sum*₁ and *sum*₂, to 0
- 1827 2. Loop through the bytes of *K* from first to last.
- 1828 3. Set *sum*₁ = *sum*₁ + the key byte (treat the key byte as a number in the range 0-255).
- 1829 4. Set *sum*₂ = *sum*₂ + *sum*₁.
- 1830 5. Encrypt *K* with *W* in ECB mode, obtaining an encrypted key, *E*.
- 1831 6. Concatenate the last 6 bytes of *E* with *sum*₂, representing *sum*₂ most-significant bit first. The
1832 result is an 8-byte block, *T*
- 1833 7. Encrypt *T* with *W* in ECB mode, obtaining an encrypted checksum, *C*.
- 1834 8. Concatenate *E* with the last 2 bytes of *C* to obtain the wrapped key.

1835 When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly,
1836 an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity
1837 bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.

1838

1839 **3 PKCS #11 Implementation Conformance**

1840 An implementation is a conforming implementation if it meets the conditions specified in one or more
1841 server profiles specified in **[PKCS #11-Prof]**.

1842 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1843 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL
1844 conform to all normative statements within the clauses specified for that profile and for any subclauses to
1845 each of those clauses .

1846 Appendix A. Acknowledgments

1847 The following individuals have participated in the creation of this specification and are gratefully
1848 acknowledged:

1849

1850 **Participants:**

1851 Gil Abel, Athena Smartcard Solutions, Inc.

1852 Warren Armstrong, QuintessenceLabs

1853 Jeff Bartell, Semper ~~Foris~~Fortis Solutions LLC

1854 Peter Bartok, Venafi, Inc.

1855 Anthony Berglas, Cryptsoft

1856 Joseph Brand, Semper Fortis Solutions LLC

1857 Kelley Burgin, National Security Agency

1858 Robert Burns, Thales e-Security

1859 Wan-Teh Chang, Google Inc.

1860 Hai-May Chao, Oracle

1861 Janice Cheng, Vormetric, Inc.

1862 Sangrae Cho, Electronics and Telecommunications Research Institute (ETRI)

1863 Doron Cohen, SafeNet, Inc.

1864 Fadi Cotran, Futurex

1865 Tony Cox, Cryptsoft

1866 Christopher Duane, EMC

1867 Chris Dunn, SafeNet, Inc.

1868 Valerie Fenwick, Oracle

1869 Terry Fletcher, SafeNet, Inc.

1870 Susan Gleeson, Oracle

1871 Sven Gossel, Charismathics

1872 John Green, QuintessenceLabs

1873 Robert Griffin, EMC

1874 Paul Grojean, Individual

1875 Peter Gutmann, Individual

1876 Dennis E. Hamilton, Individual

1877 Thomas Hardjono, M.I.T.

1878 Tim Hudson, Cryptsoft

1879 Gershon Janssen, Individual

1880 Seunghun Jin, Electronics and Telecommunications Research Institute (ETRI)

1881 Wang Jingman, Feitan Technologies

1882 Andrey Jivsov, Symantec Corp.

1883 Mark Joseph, P6R

1884 Stefan Kaesar, Infineon Technologies

1885 Greg Kazmierczak, Wave Systems Corp.

1886 Mark Knight, Thales e-Security
1887 Darren Krahn, Google Inc.
1888 Alex Krasnov, Infineon Technologies AG
1889 Dina Kurktchi-Nimeh, Oracle
1890 Mark Lambiase, SecureAuth Corporation
1891 Lawrence Lee, GoTrust Technology Inc.
1892 John Leiseboer, QuintessenceLabs
1893 Sean Leon, Infineon Technologies
1894 Geoffrey Li, Infineon Technologies
1895 Howie Liu, Infineon Technologies
1896 Hal Lockhart, Oracle
1897 Robert Lockhart, Thales e-Security
1898 Dale Moberg, Axway Software
1899 Darren Moffat, Oracle
1900 Valery Osheter, SafeNet, Inc.
1901 Sean Parkinson, EMC
1902 Rob Philpott, EMC
1903 Mark Powers, Oracle
1904 Ajai Puri, SafeNet, Inc.
1905 Robert Relyea, Red Hat
1906 Saikat Saha, Oracle
1907 Subhash Sankuratipati, NetApp
1908 Anthony Scarpino, Oracle
1909 Johann Schoetz, Infineon Technologies AG
1910 Rayees Shamsuddin, Wave Systems Corp.
1911 Radhika Siravara, Oracle
1912 Brian Smith, Mozilla Corporation
1913 David Smith, Venafi, Inc.
1914 Ryan Smith, Futurex
1915 Jerry Smith, US Department of Defense (DoD)
1916 Oscar So, Oracle
1917 Graham Steel, Cryptosense
1918 Michael Stevens, QuintessenceLabs
1919 Michael StJohns, Individual
1920 Jim Susoy, P6R
1921 Sander Temme, Thales e-Security
1922 Kiran Thota, VMware, Inc.
1923 Walter-John Turnes, Gemini Security Solutions, Inc.
1924 Stef Walter, Red Hat
1925 James Wang, Vormetric
1926 Jeff Webb, Dell
1927 Peng Yu, Feitian Technologies

- 1928 Magda Zdunkiewicz, Cryptsoft
- 1929 Chris Zimman, Individual

1930

Appendix B. Manifest constants

1931 The following constants have been defined for PKCS #11 V2.40. Also, refer to **[PKCS #11-Base]** and
1932 **[PKCS #11-Curr]** for additional definitions.

```
1933 /*  
1934 * Copyright OASIS Open 2014. All rights reserved.  
1935 * OASIS trademark, IPR and other policies apply.  
1936 * http://www.oasis-open.org/policies-guidelines/ipr  
1937 */  
1938  
1939 #define CKK_KEA 0x00000005  
1940 #define CKK_RC2 0x00000011  
1941 #define CKK_RC4 0x00000012  
1942 #define CKK_DES 0x00000013  
1943 #define CKK_CAST 0x00000016  
1944 #define CKK_CAST3 0x00000017  
1945 #define CKK_CAST5 0x00000018  
1946 #define CKK_CAST128 0x00000018  
1947 #define CKK_RC5 0x00000019  
1948 #define CKK_IDEA 0x0000001A  
1949 #define CKK_SKIPJACK 0x0000001B  
1950 #define CKK_BATON 0x0000001C  
1951 #define CKK_JUNIPER 0x0000001D  
1952 #define CKM_MD2_RSA_PKCS 0x00000004  
1953 #define CKM_MD5_RSA_PKCS 0x00000005  
1954 #define CKM_RIPEMD128_RSA_PKCS 0x00000007  
1955 #define CKM_RIPEMD160_RSA_PKCS 0x00000008  
1956 #define CKM_RC2_KEY_GEN 0x00000100  
1957 #define CKM_RC2_ECB 0x00000101  
1958 #define CKM_RC2_CBC 0x00000102  
1959 #define CKM_RC2_MAC 0x00000103  
1960 #define CKM_RC2_MAC_GENERAL 0x00000104  
1961 #define CKM_RC2_CBC_PAD 0x00000105  
1962 #define CKM_RC4_KEY_GEN 0x00000110  
1963 #define CKM_RC4 0x00000111  
1964 #define CKM_DES_KEY_GEN 0x00000120  
1965 #define CKM_DES_ECB 0x00000121  
1966 #define CKM_DES_CBC 0x00000122  
1967 #define CKM_DES_MAC 0x00000123  
1968 #define CKM_DES_MAC_GENERAL 0x00000124  
1969 #define CKM_DES_CBC_PAD 0x00000125  
1970 #define CKM_MD2 0x00000200  
1971 #define CKM_MD2_HMAC 0x00000201  
1972 #define CKM_MD2_HMAC_GENERAL 0x00000202  
1973 #define CKM_MD5 0x00000210  
1974 #define CKM_MD5_HMAC 0x00000211  
1975 #define CKM_MD5_HMAC_GENERAL 0x00000212  
1976 #define CKM_RIPEMD128 0x00000230  
1977 #define CKM_RIPEMD128_HMAC 0x00000231  
1978 #define CKM_RIPEMD128_HMAC_GENERAL 0x00000232  
1979 #define CKM_RIPEMD160 0x00000240  
1980 #define CKM_RIPEMD160_HMAC 0x00000241  
1981 #define CKM_RIPEMD160_HMAC_GENERAL 0x00000242  
1982 #define CKM_CAST_KEY_GEN 0x00000300  
1983 #define CKM_CAST_ECB 0x00000301  
1984 #define CKM_CAST_CBC 0x00000302  
1985 #define CKM_CAST_MAC 0x00000303  
1986 #define CKM_CAST_MAC_GENERAL 0x00000304  
1987 #define CKM_CAST_CBC_PAD 0x00000305  
1988 #define CKM_CAST3_KEY_GEN 0x00000310
```

1989 ~~#define CKM_CAST3_ECB 0x00000311~~
1990 ~~#define CKM_CAST3_CBC 0x00000312~~
1991 ~~#define CKM_CAST3_MAC 0x00000313~~
1992 ~~#define CKM_CAST3_MAC_GENERAL 0x00000314~~
1993 ~~#define CKM_CAST3_CBC_PAD 0x00000315~~
1994 ~~#define CKM_CAST5_KEY_GEN 0x00000320~~
1995 ~~#define CKM_CAST128_KEY_GEN 0x00000320~~
1996 ~~#define CKM_CAST5_ECB 0x00000321~~
1997 ~~#define CKM_CAST128_ECB 0x00000321~~
1998 ~~#define CKM_CAST5_CBC 0x00000322~~
1999 ~~#define CKM_CAST128_CBC 0x00000322~~
2000 ~~#define CKM_CAST5_MAC 0x00000323~~
2001 ~~#define CKM_CAST128_MAC 0x00000323~~
2002 ~~#define CKM_CAST5_MAC_GENERAL 0x00000324~~
2003 ~~#define CKM_CAST128_MAC_GENERAL 0x00000324~~
2004 ~~#define CKM_CAST5_CBC_PAD 0x00000325~~
2005 ~~#define CKM_CAST128_CBC_PAD 0x00000325~~
2006 ~~#define CKM_RC5_KEY_GEN 0x00000330~~
2007 ~~#define CKM_RC5_ECB 0x00000331~~
2008 ~~#define CKM_RC5_CBC 0x00000332~~
2009 ~~#define CKM_RC5_MAC 0x00000333~~
2010 ~~#define CKM_RC5_MAC_GENERAL 0x00000334~~
2011 ~~#define CKM_RC5_CBC_PAD 0x00000335~~
2012 ~~#define CKM_IDEA_KEY_GEN 0x00000340~~
2013 ~~#define CKM_IDEA_ECB 0x00000341~~
2014 ~~#define CKM_IDEA_CBC 0x00000342~~
2015 ~~#define CKM_IDEA_MAC 0x00000343~~
2016 ~~#define CKM_IDEA_MAC_GENERAL 0x00000344~~
2017 ~~#define CKM_IDEA_CBC_PAD 0x00000345~~
2018 ~~#define CKM_MD5_KEY_DERIVATION 0x00000390~~
2019 ~~#define CKM_MD2_KEY_DERIVATION 0x00000391~~
2020 ~~#define CKM_PBE_MD2_DES_CBC 0x000003A0~~
2021 ~~#define CKM_PBE_MD5_DES_CBC 0x000003A1~~
2022 ~~#define CKM_PBE_MD5_CAST_CBC 0x000003A2~~
2023 ~~#define CKM_PBE_MD5_CAST3_CBC 0x000003A3~~
2024 ~~#define CKM_PBE_MD5_CAST5_CBC 0x000003A4~~
2025 ~~#define CKM_PBE_MD5_CAST128_CBC 0x000003A4~~
2026 ~~#define CKM_PBE_SHA1_CAST5_CBC 0x000003A5~~
2027 ~~#define CKM_PBE_SHA1_CAST128_CBC 0x000003A5~~
2028 ~~#define CKM_PBE_SHA1_RC4_128 0x000003A6~~
2029 ~~#define CKM_PBE_SHA1_RC4_40 0x000003A7~~
2030 ~~#define CKM_PBE_SHA1_RC2_128_CBC 0x000003AA~~
2031 ~~#define CKM_PBE_SHA1_RC2_40_CBC 0x000003AB~~
2032 ~~#define CKM_KEY_WRAP_LYNKS 0x00000400~~
2033 ~~#define CKM_KEY_WRAP_SET_OAEP 0x00000401~~
2034 ~~#define CKM_SKIPJACK_KEY_GEN 0x00001000~~
2035 ~~#define CKM_SKIPJACK_ECB64 0x00001001~~
2036 ~~#define CKM_SKIPJACK_CBC64 0x00001002~~
2037 ~~#define CKM_SKIPJACK_OFB64 0x00001003~~
2038 ~~#define CKM_SKIPJACK_CFB64 0x00001004~~
2039 ~~#define CKM_SKIPJACK_CFB32 0x00001005~~
2040 ~~#define CKM_SKIPJACK_CFB16 0x00001006~~
2041 ~~#define CKM_SKIPJACK_CFB8 0x00001007~~
2042 ~~#define CKM_SKIPJACK_WRAP 0x00001008~~
2043 ~~#define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009~~
2044 ~~#define CKM_SKIPJACK_RELAYX 0x0000100a~~
2045 ~~#define CKM_KEA_KEY_PAIR_GEN 0x00001010~~
2046 ~~#define CKM_KEA_KEY_DERIVE 0x00001011~~
2047 ~~#define CKM_FORTEZZA_TIMESTAMP 0x00001020~~
2048 ~~#define CKM_BATON_KEY_GEN 0x00001030~~
2049 ~~#define CKM_BATON_ECB128 0x00001031~~
2050 ~~#define CKM_BATON_ECB96 0x00001032~~
2051 ~~#define CKM_BATON_CBC128 0x00001033~~
2052 ~~#define CKM_BATON_COUNTER 0x00001034~~

2053 ~~#define CKM_BATON_SHUFFLE 0x00001035~~
2054 ~~#define CKM_BATON_WRAP 0x00001036~~
2055 ~~#define CKM_JUNIPER_KEY_GEN 0x00001060~~
2056 ~~#define CKM_JUNIPER_ECB128 0x00001061~~
2057 ~~#define CKM_JUNIPER_CBC128 0x00001062~~
2058 ~~#define CKM_JUNIPER_COUNTER 0x00001063~~
2059 ~~#define CKM_JUNIPER_SHUFFLE 0x00001064~~
2060 ~~#define CKM_JUNIPER_WRAP 0x00001065~~
2061 ~~#define CKM_FASTHASH 0x00001070~~

2062

2063

2064 The definitions for manifest constants specified in this document can be found in the following normative
2065 computer language definition files:

2066 • [include/pkcs11-v2.40/pkcs11.h](#)

2067 • [include/pkcs11-v2.40/pkcs11t.h](#)

2068 • [include/pkcs11-v2.40/pkcs11f.h](#)

2069 These files are linked from the Related Work section at the top of this specification.

2070

Appendix C. Revision History

2071

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 2014	Robert Griffin	Updated revision history and participant list in preparation for Committee Specification ballot
wd07	Nov 3 2014	Robert Griffin	Editorial corrections
os	Apr 14 2015	OASIS	OASIS Standard
os-rev01	Dec 9 2015	Robert Griffin / Tim Hudson	Change bar edits corresponding to Errata01

2072