



PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40

Committee Specification Draft 02

08 April 2014

Specification URIs

This version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd02/pkcs11-hist-v2.40-csd02.doc>

(Authoritative)

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd02/pkcs11-hist-v2.40-csd02.html>

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd02/pkcs11-hist-v2.40-csd02.pdf>

Previous version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.doc>

(Authoritative)

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.html>

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.pdf>

Latest version:

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

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.pdf>

Technical Committee:

OASIS PKCS 11 TC

Chairs:

Robert Griffin (robert.griffin@rsa.com), EMC Corporation

Valerie Fenwick (valerie.fenwick@oracle.com), Oracle

Editors:

Susan Gleeson (susan.gleeson@oracle.com), Oracle

Chris Zimman (czimman@bloomberg.com), Bloomberg Finance L.P.

Related work:

This specification is related to:

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

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

Abstract:

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

Status:

This document was last revised or approved by the OASIS PKCS 11 TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at <https://www.oasis-open.org/committees/pkcs11/>.

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

Citation format:

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

[PKCS11-hist-v2.40]

PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40. Edited by Susan Gleeson and Chris Zimman. 08 April 2014. OASIS Committee Specification Draft 02. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd02/pkcs11-hist-v2.40-csd02.html>. Latest version: <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>.

Notices

Copyright © OASIS Open 2014. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

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

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

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

2.18.1	Definitions	59
2.18.2	LYNKS key wrapping	59
3	PKCS #11 Implementation Conformance	60
Appendix A.	Acknowledgments	61
Appendix B.	Manifest constants	63
Appendix C.	Revision History	66

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.

38	PRF	Pseudo random function.
39	RSA	The RSA public-key cryptosystem.
40	RC2	RSA Security's RC2 symmetric block cipher.
41	RC4	RSA Security's proprietary RC4 symmetric stream cipher.
42	RC5	RSA Security's RC5 symmetric block cipher.
43	SET	The Secure Electronic Transaction protocol.
44	SHA-1	The (revised) Secure Hash Algorithm with a 160-bit message digest, as defined in FIPS PUB 180-2.
46	SKIPJACK	MISSI's SKIPJACK block cipher.
47		

48 1.4 Normative References

49	[PKCS #11-Base]	<i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i> Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html .
50		
51		
52		
53	[PKCS #11-Curr]	<i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40.</i> Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html .
54		
55		
56		
57	[PKCS #11-Prof]	<i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i> Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html .
58		
59		
60		
61	[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 .
62		
63		

64 1.5 Non-Normative References

65	[ANSI C]	ANSI/ISO. <i>American National Standard for Programming Languages – C.</i> 1990
66	[ANSI X9.31]	Accredited Standards Committee X9. <i>Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA).</i> 1998.
67		
68	[ANSI X9.42]	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.</i> 2003
69		
70		
71	[ANSI X9.62]	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).</i> 1998
72		
73	[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/
74		
75		
76		
77	[CDPD]	Ameritech Mobile Communications et al. <i>Cellular Digital Packet Data System Specifications: Part 406: Airlink Security.</i> 1993
78		
79	[FIPS PUB 46-3]	NIST. <i>FIPS 46-3: Data Encryption Standard (DES).</i> October 26, 2009. URL: http://csrc.nist.gov/publications/fips/index.html
80		
81	[FIPS PUB 81]	NIST. <i>FIPS 81: DES Modes of Operation.</i> December 1980. URL: http://csrc.nist.gov/publications/fips/index.html
82		

- 83 [FIPS PUB 113] NIST. *FIPS 113: Computer Data Authentication*. May 30, 1985. URL:
84 <http://csrc.nist.gov/publications/fips/index.html>
- 85 [FIPS PUB 180-2] NIST. *FIPS 180-2: Secure Hash Standard*. August 1, 2002. URL:
86 <http://csrc.nist.gov/publications/fips/index.html>
- 87 [FORTEZZA CIPG] NSA, Workstation Security Products. *FORTEZZA Cryptologic Interface*
88 *Programmers Guide, Revision 1.52*. November 1985
- 89 [GCS-API] X/Open Company Ltd. *Generic Cryptographic Service API (GCS-API), Base –*
90 *Draft 2*. February 14, 1995.
- 91 [ISO/IEC 7816-1] ISO/IEC 7816-1:2011. *Identification Cards – Integrated circuit cards -- Part 1:*
92 *Cards with contacts -- Physical Characteristics*. 2011 URL:
93 http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089.
- 94 [ISO/IEC 7816-4] ISO/IEC 7618-4:2013. *Identification Cards – Integrated circuit cards – Part 4:*
95 *Organization, security and commands for interchange*. 2013. URL:
96 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54550.
- 97 [ISO/IEC 8824-1] ISO/IEC 8824-1:2008. *Abstract Syntax Notation One (ASN.1): Specification of*
98 *Base Notation*. 2002. URL:
99 http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012
- 100 [ISO/IEC 8825-1] ISO/IEC 8825-1:2008. *Information Technology – ASN.1 Encoding Rules:*
101 *Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER),*
102 *and Distinguished Encoding Rules (DER)*. 2008. URL:
103 http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60
- 104 [ISO/IEC 9594-1] ISO/IEC 9594-1:2008. *Information Technology – Open System Interconnection –*
105 *The Directory: Overview of Concepts, Models and Services*. 2008. URL:
106 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53364
- 107 [ISO/IEC 9594-8] ISO/IEC 9594-8:2008. *Information Technology – Open Systems Interconnection*
108 *– The Directory: Public-key and Attribute Certificate Frameworks*. 2008 URL:
109 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53372
- 110 [ISO/IEC 9796-2] ISO/IEC 9796-2:2010. *Information Technology – Security Techniques – Digital*
111 *Signature Scheme Giving Message Recovery – Part 2: Integer factorization*
112 *based mechanisms*. 2010. URL:
113 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788
- 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

136	[PKCS #11-UG]	<i>PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40</i> . Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html .
137		
138		
139	[PKCS #12]	RSA Laboratories. <i>Personal Information Exchange Syntax Standard, v1.0</i> . June 1999. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf
140		
141	[RFC 1321]	R. Rivest. <i>RFC 1321: The MD5 Message-Digest Algorithm</i> . MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. URL: http://www.rfc-editor.org/rfc/rfc1321.txt
142		
143		
144	[RFC 3369]	R. Houseley. <i>RFC 3369: Cryptographic Message Syntax (CMS)</i> . August 2002. URL: http://www.rfc-editor.org/rfc/rfc3369.txt
145		
146	[RFC 6149]	S. Turner and L. Chen. <i>RFC 6149: MD2 to Historic Status</i> . March, 2011. URL: http://www.rfc-editor.org/rfc/rfc6149.txt
147		
148	[SEC-1]	Standards for Efficient Cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography</i> . Version 1.0, September 20, 2000.
149		
150		
151	[SEC-2]	Standards for Efficient cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters</i> . Version 1.0, September 20, 2000.
152		
153		
154	[TLS]	IETF. <i>RFC 2246: The TLS Protocol Version 1.0</i> . January 1999. URL: http://ietf.org/rfc/rfc2256.txt
155		
156	[WIM]	WAP. <i>Wireless Identity Module</i> . – WAP-260-WIP-20010712.a. July 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf
157		
158		
159	[WPKI]	WAP. <i>Wireless Application Protocol: Public Key Infrastructure Definition</i> . – WAP-217-WPKI-20010424-a. April 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf
160		
161		
162		
163	[WTLS]	WAP. <i>Wireless Transport Layer Security Version</i> – WAP-261-WTLS-20010406-a. April 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf
164		
165		
166		
167	[X.500]	ITU-T. <i>Information Technology – Open Systems Interconnection – The Directory: Overview of Concepts, Models and Services</i> . February 2001. (Identical to ISO/IEC 9594-1)
168		
169		
170	[X.509]	ITU-T. <i>Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks</i> . March 2000. (Identical to ISO/IEC 9594-8)
171		
172		
173	[X.680]	ITU-T. <i>Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation</i> . July 2002. (Identical to ISO/IEC 8824-1)
174		
175	[X.690]	ITU-T. <i>Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)</i> . July 2002. (Identical to ISO/IEC 8825-1)
176		
177		
178		

179

2 Mechanisms

180

2.1 PKCS #11 Mechanisms

181

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

182

183

184

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

185

186

187

188

189

190

191

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	

192 ¹ SR = SignRecover, VR = VerifyRecover.

193 ² Single-part operations only.

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

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

197 In general, if a mechanism makes no mention of the *ulMinKeyLen* and *ulMaxKeyLen* fields of the
198 CK_MECHANISM_INFO structure, then those fields have no meaning for that particular mechanism.
199

200 2.2 FORTEZZA timestamp

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

204 **It has no parameters.**

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

207 *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

208 ¹ Single-part operations only

209 ² Data length, signature length

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

212 2.3 KEA

213 2.3.1 Definitions

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

216 Mechanisms:

217 CKM_KEA_KEY_PAIR_GEN

218 CKM_KEA_KEY_DERIVE

219 2.3.2 KEA mechanism parameters

220 2.3.2.1 CK_KEA_DERIVE_PARAMS; CK_KEA_DERIVE_PARAMS_PTR

221

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

```
224 typedef struct CK_KEA_DERIVE_PARAMS {  
225     CK_BBOOL isSender;  
226     CK_ULONG ulRandomLen;  
227     CK_BYTE_PTR pRandomA;  
228     CK_BYTE_PTR pRandomB;  
229     CK_ULONG ulPublicDataLen;  
230     CK_BYTE_PTR pPublicData;  
231 } CK_KEA_DERIVE_PARAMS;
```

232

233 The fields of the structure have the following meanings:

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

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

238 *pRandomA* pointer to Ra data

239 *pRandomB* pointer to Rb data

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

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

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

243 2.3.3 KEA public key objects

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

247 *Table 3, KEA Public Key Object Attributes*

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

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

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

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

```

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

269

270 2.3.4 KEA private key objects

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

274 *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

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

276

277 The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the “KEA domain
278 parameters”.

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

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

```
283 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;  
284 CK_KEY_TYPE keyType = CKK_KEA;  
285 CK_UTF8CHAR label[] = "A KEA private key object";  
286 CK_BYTE subject[] = {...};  
287 CK_BYTE id[] = {123};  
288 CK_BYTE prime[] = {...};  
289 CK_BYTE subprime[] = {...};  
290 CK_BYTE base[] = {...};  
291 CK_BYTE value[] = {...};  
292 CK_BBOOL true = CK_TRUE;  
293 CK_ATTRIBUTE template[] = {  
294     {CKA_CLASS, &class, sizeof(class)},  
295     {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS  
296     {CKA_TOKEN, &>true, sizeof(true)},  
297     {CKA_LABEL, label, sizeof(label) - 1},  
298     {CKA_SUBJECT, subject, sizeof(subject)},  
299     {CKA_ID, id, sizeof(id)},  
300     {CKA_SENSITIVE, &true, sizeof(true)},  
301     {CKA_DERIVE, &true, sizeof(true)},  
302     {CKA_PRIME, prime, sizeof(prime)},  
303     {CKA_SUBPRIME, subprime, sizeof(subprime)},  
304     {CKA_BASE, base, sizeof(base)},  
305     {CKA_VALUE, value, sizeof(value)}  
306 };
```

307 2.3.5 KEA key pair generation

308 The KEA key pair generation mechanism, denoted **CKM_KEA_KEY_PAIR_GEN**, generates key pairs for
309 the Key Exchange Algorithm, as defined by NIST’s “SKIPJACK and KEA Algorithm Specification Version
310 2.0”, 29 May 1998.

311 It does not have a parameter.

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

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

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

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

323 **2.3.6 KEA key derivation**

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

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

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

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

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

343 *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.

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

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

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

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

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

362 2.4 RC2

363 2.4.1 Definitions

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

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

371 Mechanisms:

- 372 CKM_RC2_KEY_GEN
- 373 CKM_RC2_ECB
- 374 CKM_RC2_CBC
- 375 CKM_RC2_MAC
- 376 CKM_RC2_MAC_GENERAL
- 377 CKM_RC2_CBC_PAD

378 2.4.2 RC2 secret key objects

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

382 *Table 6, RC2 Secret Key Object Attributes*

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

* Note that the rules regarding the **CKA_SENSITIVE**, **CKA_EXTRACTABLE**,
CKA_ALWAYS_SENSITIVE, and **CKA_NEVER_EXTRACTABLE** attributes have changed in version
2.11 to match the policy used by other key derivation mechanisms such as
CKM_SSL3_MASTER_KEY_DERIVE.

CKA_VALUE ^{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

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

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

```

385 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
386 CK_KEY_TYPE keyType = CKK_RC2;
387 CK_UTF8CHAR label[] = "An RC2 secret key object";
388 CK_BYTE value[] = {...};
389 CK_BBOOL true = CK_TRUE;
390 CK_ATTRIBUTE template[] = {
391     {CKA_CLASS, &class, sizeof(class)},
392     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
393     {CKA_TOKEN, &>true, sizeof(true)},
394     {CKA_LABEL, label, sizeof(label)-1},
395     {CKA_ENCRYPT, &>true, sizeof(true)},
396     {CKA_VALUE, value, sizeof(value)}
397 };

```

398 2.4.3 RC2 mechanism parameters

399 2.4.3.1 CK_RC2_PARAMS; CK_RC2_PARAMS_PTR

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

```

402 typedef CK_ULONG CK_RC2_PARAMS;

```

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

404 2.4.3.2 CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR

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

```

407 typedef struct CK_RC2_CBC_PARAMS {
408     CK_ULONG ulEffectiveBits;
409     CK_BYTE iv[8];
410 } CK_RC2_CBC_PARAMS;

```

411 The fields of the structure have the following meanings:

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

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

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

416 2.4.3.3 CK_RC2_MAC_GENERAL_PARAMS; 417 CK_RC2_MAC_GENERAL_PARAMS_PTR

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

```

420 typedef struct CK_RC2_MAC_GENERAL_PARAMS {
421     CK_ULONG ulEffectiveBits;
422     CK_ULONG ulMacLength;
423 } CK_RC2_MAC_GENERAL_PARAMS;

```

424 The fields of the structure have the following meanings:
 425 *ulEffectiveBits* the effective number of bits in the RC2 search space
 426 *ulMacLength* length of the MAC produced, in bytes
 427 **CK_RC2_MAC_GENERAL_PARAMS_PTR** is a pointer to a **CK_RC2_MAC_GENERAL_PARAMS**.

428 2.4.4 RC2 key generation

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

431 It does not have a parameter.

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

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

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

439 2.4.5 RC2-ECB

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

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

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

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

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

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

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

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

459 **2.4.6 RC2-CBC**

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

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

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

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

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

477 *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	

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

480 **2.4.7 RC2-CBC with PKCS padding**

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

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

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

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

493 for data length constraints when wrapping and unwrapping keys do not apply to wrapping and
494 unwrapping private keys.

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

496 *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

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

499 2.4.8 General-length RC2-MAC

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

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

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

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

508 *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

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

511 2.4.9 RC2-MAC

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

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

517

518 *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

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

521 2.5 RC4

522 2.5.1 Definitions

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

525 Mechanisms

526 CKM_RC4_KEY_GEN

527 CKM_RC4

528 2.5.2 RC4 secret key objects

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

532 *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

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

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

```
535 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
536 CK_KEY_TYPE keyType = CKK_RC4;  
537 CK_UTF8CHAR label[] = "An RC4 secret key object";  
538 CK_BYTE value[] = {...};  
539 CK_BBOOL true = CK_TRUE;  
540 CK_ATTRIBUTE template[] = {  
541     {CKA_CLASS, &class, sizeof(class)},  
542     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
543     {CKA_TOKEN, &>true, sizeof(true)},  
544     {CKA_LABEL, label, sizeof(label)-1},  
545     {CKA_ENCRYPT, &>true, sizeof(true)},  
546     {CKA_VALUE, value, sizeof(value)}  
547 };
```

548 2.5.3 RC4 key generation

549 The RC4 key generation mechanism, denoted **CKM_RC4_KEY_GEN**, is a key generation mechanism for
550 RSA Security's proprietary stream cipher RC4.

551 It does not have a parameter.

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

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

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

559 2.5.4 RC4 mechanism

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

562 It does not have a parameter.

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

564 *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

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

567 2.6 RC5

568 2.6.1 Definitions

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

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

573 Mechanisms:

574 CKM_RC5_KEY_GEN

575 CKM_RC5_ECB

576 CKM_RC5_CBC

577 CKM_RC5_MAC

578 CKM_RC5_MAC_GENERAL

579 CMK_RC5_CBC_PAD

580 2.6.2 RC5 secret key objects

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

584 *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

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

586

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

```
588 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
589 CK_KEY_TYPE keyType = CKK_RC5;  
590 CK_UTF8CHAR label[] = "An RC5 secret key object";  
591 CK_BYTE value[] = {...};  
592 CK_BBOOL true = CK_TRUE;
```

```

593 CK_ATTRIBUTE template[] = {
594     {CKA_CLASS, &class, sizeof(class)},
595     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
596     {CKA_TOKEN, &true, sizeof(true)},
597     {CKA_LABEL, label, sizeof(label)-1},
598     {CKA_ENCRYPT, &true, sizeof(true)},
599     {CKA_VALUE, value, sizeof(value)}
600 };

```

601 2.6.3 RC5 mechanism parameters

602 2.6.3.1 CK_RC5_PARAMS; CK_RC5_PARAMS_PTR

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

```

605 typedef struct CK_RC5_PARAMS {
606     CK_ULONG ulWordsize;
607     CK_ULONG ulRounds;
608 } CK_RC5_PARAMS;

```

609 The fields of the structure have the following meanings:

610 *ulWordsize* wordsize of RC5 cipher in bytes

611 *ulRounds* number of rounds of RC5 encipherment

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

613 2.6.3.2 CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR

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

```

616 typedef struct CK_RC5_CBC_PARAMS {
617     CK_ULONG ulWordsize;
618     CK_ULONG ulRounds;
619     CK_BYTE_PTR pIV;
620     CK_ULONG ulIVLen;
621 } CK_RC5_CBC_PARAMS;

```

622 The fields of the structure have the following meanings:

623 *ulwordSize* wordsize of RC5 cipher in bytes

624 *ulRounds* number of rounds of RC5 encipherment

625 *pIV* pointer to initialization vector (IV) for CBC encryption

626 *ulIVLen* length of initialization vector (must be same as
627 blocksize)

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

629 2.6.3.3 CK_RC5_MAC_GENERAL_PARAMS; 630 CK_RC5_MAC_GENERAL_PARAMS_PTR

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

```

633 typedef struct CK_RC5_MAC_GENERAL_PARAMS {
634     CK_ULONG ulWordsize;
635     CK_ULONG ulRounds;
636     CK_ULONG ulMacLength;
637 } CK_RC5_MAC_GENERAL_PARAMS;

```

638 The fields of the structure have the following meanings:

639 *ulwordSize* wordsize of RC5 cipher in bytes

640 *ulRounds* number of rounds of RC5 encipherment

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

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

643 2.6.4 RC5 key generation

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

646 It does not have a parameter.

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

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

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

654 2.6.5 RC5-ECB

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

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

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

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

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

671 *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	

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

674 2.6.6 RC5-CBC

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

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

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

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

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

691 *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	

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

694 2.6.7 RC5-CBC with PKCS padding

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

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

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

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

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

709 *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

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

712 2.6.8 General-length RC5-MAC

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

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

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

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

721 *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

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

724 2.6.9 RC5-MAC

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

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

729 *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]

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

732 2.7 General block cipher

733 2.7.1 Definitions

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

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

740 Mechanisms:

741 CKM_DES_KEY_GEN
742 CKM_DES_ECB
743 CKM_DES_CBC
744 CKM_DES_MAC
745 CKM_DES_MAC_GENERAL
746 CKM_DES_CBC_PAD
747 CKM_CDMF_KEY_GEN
748 CKM_CDMF_ECB
749 CKM_CDMF_CBC
750 CKM_CDMF_MAC
751 CKM_CDMF_MAC_GENERAL
752 CKM_CDMF_CBC_PAD
753 CKM_DES_OFB64
754 CKM_DES_OFB8
755 CKM_DES_CFB64
756 CKM_DES_CFB8
757 CKM_CAST_KEY_GEN
758 CKM_CAST_ECB
759 CKM_CAST_CBC
760 CKM_CAST_MAC
761 CKM_CAST_MAC_GENERAL
762 CKM_CAST_CBC_PAD
763 CKM_CAST3_KEY_GEN
764 CKM_CAST3_ECB
765 CKM_CAST3_CBC
766 CKM_CAST3_MAC
767 CKM_CAST3_MAC_GENERAL

768 CKM_CAST3_CBC_PAD
 769 CKM_CAST5_KEY_GEN
 770 CKM_CAST128_KEY_GEN
 771 CKM_CAST5_ECB
 772 CKM_CAST128_ECB
 773 CKM_CAST5_CBC
 774 CKM_CAST128_CBC
 775 CKM_CAST5_MAC
 776 CKM_CAST128_MAC
 777 CKM_CAST5_MAC_GENERAL
 778 CKM_CAST128_MAC_GENERAL
 779 CKM_CAST5_CBC_PAD
 780 CKM_CAST128_CBC_PAD
 781 CKM_IDEA_KEY_GEN
 782 CKM_IDEA_ECB
 783 CKM_IDEA_MAC
 784 CKM_IDEA_MAC_GENERAL
 785 CKM_IDEA_CBC_PAD

786 2.7.2 DES secret key objects

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

790 *Table 20, DES Secret Key Object*

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

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

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

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

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

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

811 2.7.3 CAST secret key objects

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

815 *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

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

817

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

```
819 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
820 CK_KEY_TYPE keyType = CKK_CAST;  
821 CK_UTF8CHAR label[] = "A CAST secret key object";  
822 CK_BYTE value[] = {...};  
823 CK_BBOOL true = CK_TRUE;  
824 CK_ATTRIBUTE template[] = {  
825     {CKA_CLASS, &class, sizeof(class)},  
826     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
827     {CKA_TOKEN, &>true, sizeof(true)},  
828     {CKA_LABEL, label, sizeof(label)-1},  
829     {CKA_ENCRYPT, &>true, sizeof(true)},  
830     {CKA_VALUE, value, sizeof(value)}  
831 };
```

832 2.7.4 CAST3 secret key objects

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

836 *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

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

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

```
839 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
840 CK_KEY_TYPE keyType = CKK_CAST3;  
841 CK_UTF8CHAR label[] = "A CAST3 secret key object";  
842 CK_BYTE value[] = {...};  
843 CK_BBOOL true = CK_TRUE;  
844 CK_ATTRIBUTE template[] = {  
845     {CKA_CLASS, &class, sizeof(class)},  
846     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
847     {CKA_TOKEN, &>true, sizeof(true)},  
848     {CKA_LABEL, label, sizeof(label)-1},  
849     {CKA_ENCRYPT, &>true, sizeof(true)},  
850     {CKA_VALUE, value, sizeof(value)}  
851 };
```


852 **2.7.5 CAST128 (CAST5) secret key objects**

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

856 *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

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

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

```

859 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
860 CK_KEY_TYPE keyType = CKK_CAST128;
861 CK_UTF8CHAR label[] = "A CAST128 secret key object";
862 CK_BYTE value[] = {...};
863 CK_BBOOL true = CK_TRUE;
864 CK_ATTRIBUTE template[] = {
865     {CKA_CLASS, &class, sizeof(class)},
866     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
867     {CKA_TOKEN, &>true, sizeof(true)},
868     {CKA_LABEL, label, sizeof(label)-1},
869     {CKA_ENCRYPT, &>true, sizeof(true)},
870     {CKA_VALUE, value, sizeof(value)}
871 };

```

872

873 **2.7.6 IDEA secret key objects**

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

876 *Table 24, IDEA Secret Key Object*

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

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

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

```

879 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
880 CK_KEY_TYPE keyType = CKK_IDEA;
881 CK_UTF8CHAR label[] = "An IDEA secret key object";
882 CK_BYTE value[16] = {...};
883 CK_BBOOL true = CK_TRUE;
884 CK_ATTRIBUTE template[] = {
885     {CKA_CLASS, &class, sizeof(class)},
886     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
887     {CKA_TOKEN, &>true, sizeof(true)},
888     {CKA_LABEL, label, sizeof(label)-1},
889     {CKA_ENCRYPT, &>true, sizeof(true)},
890     {CKA_VALUE, value, sizeof(value)}
891 };

```

892

893 2.7.7 CDMF secret key objects

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

896 *Table 25, CDMF Secret Key Object*

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

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

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

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

```
902 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
903 CK_KEY_TYPE keyType = CKK_CDMF;  
904 CK_UTF8CHAR label[] = "A CDMF secret key object";  
905 CK_BYTE value[8] = {...};  
906 CK_BBOOL true = CK_TRUE;  
907 CK_ATTRIBUTE template[] = {  
908     {CKA_CLASS, &class, sizeof(class)},  
909     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
910     {CKA_TOKEN, &>true, sizeof(true)},  
911     {CKA_LABEL, label, sizeof(label)-1},  
912     {CKA_ENCRYPT, &>true, sizeof(true)},  
913     {CKA_VALUE, value, sizeof(value)}  
914 };
```

915 2.7.8 General block cipher mechanism parameters

916 2.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR

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

```
923 typedef CK_ULONG CK_MAC_GENERAL_PARAMS;  
924
```

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

926 2.7.9 General block cipher key generation

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

929 This mechanism does not have a parameter.

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

933 When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB
934 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits
935 set properly.

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

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

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

946 2.7.10 General block cipher ECB

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

950 It does not have a parameter.

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

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

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

962 *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	

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

968 2.7.11 General block cipher CBC

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

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

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

975 *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	

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

981 2.7.12 General block cipher CBC with PKCS padding

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

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

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

991

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

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

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

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

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

998 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
999 MAY be used. The CAST, CAST3 and CAST128 (CAST5) ciphers have variable key sizes, and so for
1000 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1001 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA, and CDMF
1002 ciphers, these fields are not used.

1003 2.7.13 General-length general block cipher MAC

1004 Cipher <NAME> has a general-length MACing mode, "General-length <NAME>-MAC", denoted
1005 **CKM_<NAME>_MAC_GENERAL**. It is a mechanism for single-and multiple-part signatures and
1006 verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB
1007 113.

1008 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which specifies the size of the output.

1009 The output bytes from this mechanism are taken from the start of the final cipher block produced in the
1010 MACing process.

1011 Constraints on key types and the length of input and output data are summarized in the following table:

1012 *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

1013 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1014 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for
1015 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1016 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF
1017 ciphers, these fields are not used.

1018 2.7.14 General block cipher MAC

1019 Cipher <NAME> has a MACing mechanism, "<NAME>-MAC", denoted **CKM_<NAME>_MAC**. This
1020 mechanism is a special case of the **CKM_<NAME>_MAC_GENERAL** mechanism described above. It
1021 produces an output of size half as large as <NAME>'s blocksize.

1022 This mechanism has no parameters.

1023 Constraints on key types and the length of data are summarized in the following table:

1024 *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]

1025 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1026 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for
1027 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1028 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF
1029 ciphers, these fields are not used.

1030 2.8 SKIPJACK

1031 2.8.1 Definitions

1032 This section defines the key type “CKK_SKIPJACK” for type CK_KEY_TYPE as used in the
1033 CKA_KEY_TYPE attribute of key objects.

1034 Mechanisms:

- 1035 CKM_SKIPJACK_KEY_GEN
- 1036 CKM_SKIPJACK_ECB64
- 1037 CKM_SKIPJACK_CBC64
- 1038 CKM_SKIPJACK_OFB64
- 1039 CKM_SKIPJACK_CFB64
- 1040 CKM_SKIPJACK_CFB32
- 1041 CKM_SKIPJACK_CFB16
- 1042 CKM_SKIPJACK_CFB8
- 1043 CKM_SKIPJACK_WRAP
- 1044 CKM_SKIPJACK_PRIVATE_WRAP
- 1045 CKM_SKIPJACK_RELAYX

1046 2.8.2 SKIPJACK secret key objects

1047 SKIPJACK secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_SKIPJACK**) holds a
1048 single-length MEK or a TEK. The following table defines the SKIPJACK secret object attributes, in
1049 addition to the common attributes defined for this object class:

1050 *Table 31, SKIPJACK Secret Key Object*

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

1051 Refer to [PKCS #11-Base] table 15 for footnotes

1052

1053 SKIPJACK keys have 16 checksum bits, and these bits must be properly set. Attempting to create or
1054 unwrap a SKIPJACK key with incorrect checksum bits MUST return an error.

1055 It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK
1056 key with a specified value. Nonetheless, we provide templates for doing so.

1057 The following is a sample template for creating a SKIPJACK MEK secret key object:

```
1058 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1059 CK_KEY_TYPE keyType = CKK_SKIPJACK;  
1060 CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";  
1061 CK_BYTE value[12] = {...};  
1062 CK_BBOOL true = CK_TRUE;  
1063 CK_ATTRIBUTE template[] = {  
1064     {CKA_CLASS, &class, sizeof(class)},  
1065     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1066     {CKA_TOKEN, &true, sizeof(true)},  
1067     {CKA_LABEL, label, sizeof(label)-1},  
1068     {CKA_ENCRYPT, &true, sizeof(true)},  
1069     {CKA_VALUE, value, sizeof(value)}  
1070 };
```

1071 The following is a sample template for creating a SKIPJACK TEK secret key object:


```

1072 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1073 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1074 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1075 CK_BYTE value[12] = {...};
1076 CK_BBOOL true = CK_TRUE;
1077 CK_ATTRIBUTE template[] = {
1078     {CKA_CLASS, &class, sizeof(class)},
1079     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1080     {CKA_TOKEN, &true, sizeof(true)},
1081     {CKA_LABEL, label, sizeof(label)-1},
1082     {CKA_ENCRYPT, &true, sizeof(true)},
1083     {CKA_WRAP, &true, sizeof(true)},
1084     {CKA_VALUE, value, sizeof(value)}
1085 };

```

1086 2.8.3 SKIPJACK Mechanism parameters

1087 2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; 1088 CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR

1089 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS** is a structure that provides the parameters to the
1090 **CKM_SKIPJACK_PRIVATE_WRAP** mechanism. It is defined as follows:

```

1091 typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
1092     CK_ULONG ulPasswordLen;
1093     CK_BYTE_PTR pPassword;
1094     CK_ULONG ulPublicDataLen;
1095     CK_BYTE_PTR pPublicData;
1096     CK_ULONG ulPandGLen;
1097     CK_ULONG ulQLen;
1098     CK_ULONG ulRandomLen;
1099     CK_BYTE_PTR pRandomA;
1100     CK_BYTE_PTR pPrimeP;
1101     CK_BYTE_PTR pBaseG;
1102     CK_BYTE_PTR pSubprimeQ;
1103 } CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

```

1104 The fields of the structure have the following meanings:

1105	<i>ulPasswordLen</i>	length of the password
1106	<i>pPassword</i>	pointer to the buffer which contains the user-supplied password
1107		
1108	<i>ulPublicDataLen</i>	other party's key exchange public key size
1109	<i>pPublicData</i>	pointer to other party's key exchange public key value
1110	<i>ulPandGLen</i>	length of prime and base values
1111	<i>ulQLen</i>	length of subprime value
1112	<i>ulRandomLen</i>	size of random Ra, in bytes
1113	<i>pPrimeP</i>	pointer to Prime, p, value
1114	<i>pBaseG</i>	pointer to Base, b, value

1115 *pSubprimeQ* pointer to Subprime, q, value

1116 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR** is a pointer to a
1117 **CK_PRIVATE_WRAP_PARAMS**.

1118 **2.8.3.2 CK_SKIPJACK_RELAYX_PARAMS;** 1119 **CK_SKIPJACK_RELAYX_PARAMS_PTR**

1120 **CK_SKIPJACK_RELAYX_PARAMS** is a structure that provides the parameters to the
1121 **CKM_SKIPJACK_RELAYX** mechanism. It is defined as follows:

```
1122 typedef struct CK_SKIPJACK_RELAYX_PARAMS {  
1123     CK_ULONG ulOldWrappedXLen;  
1124     CK_BYTE_PTR pOldWrappedX;  
1125     CK_ULONG ulOldPasswordLen;  
1126     CK_BYTE_PTR pOldPassword;  
1127     CK_ULONG ulOldPublicDataLen;  
1128     CK_BYTE_PTR pOldPublicData;  
1129     CK_ULONG ulOldRandomLen;  
1130     CK_BYTE_PTR pOldRandomA;  
1131     CK_ULONG ulNewPasswordLen;  
1132     CK_BYTE_PTR pNewPassword;  
1133     CK_ULONG ulNewPublicDataLen;  
1134     CK_BYTE_PTR pNewPublicData;  
1135     CK_ULONG ulNewRandomLen;  
1136     CK_BYTE_PTR pNewRandomA;  
1137 } CK_SKIPJACK_RELAYX_PARAMS;
```

1138 The fields of the structure have the following meanings:

1139 *ulOldWrappedLen* length of old wrapped key in bytes

1140 *pOldWrappedX* pointer to old wrapper key

1141 *ulOldPasswordLen* length of the old password

1142 *pOldPassword* pointer to the buffer which contains the old user-supplied
1143 password

1144 *ulOldPublicDataLen* old key exchange public key size

1145 *pOldPublicData* pointer to old key exchange public key value

1146 *ulOldRandomLen* size of old random Ra in bytes

1147 *pOldRandomA* pointer to old Ra data

1148 *ulNewPasswordLen* length of the new password

1149 *pNewPassword* pointer to the buffer which contains the new user-
1150 supplied password

1151 *ulNewPublicDataLen* new key exchange public key size

1152 *pNewPublicData* pointer to new key exchange public key value

1153 *ulNewRandomLen* size of new random Ra in bytes

1154 *pNewRandomA* pointer to new Ra data

1155 **CK_SKIPJACK_RELAYX_PARAMS_PTR** is a pointer to a **CK_SKIPJACK_RELAYX_PARAMS**.

1156 2.8.4 SKIPJACK key generation

1157 The SKIPJACK key generation mechanism, denoted **CKM_SKIPJACK_KEY_GEN**, is a key generation
1158 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1159 It does not have a parameter.

1160 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
1161 key.

1162 2.8.5 SKIPJACK-ECB64

1163 SKIPJACK-ECB64, denoted **CKM_SKIPJACK_ECB64**, is a mechanism for single- and multiple-part
1164 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB
1165 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 cant 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 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

1171 2.8.6 SKIPJACK-CBC64

1172 SKIPJACK-CBC64, denoted **CKM_SKIPJACK_CBC64**, 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 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

1179 2.8.7 SKIPJACK-OFB64

1180 SKIPJACK-OFB64, denoted **CKM_SKIPJACK_OFB64**, is a mechanism for single- and multiple-part
1181 encryption and decryption with SKIPJACK in 64-bit output 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 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

1187 2.8.8 SKIPJACK-CFB64

1188 SKIPJACK-CFB64, denoted **CKM_SKIPJACK_CFB64**, is a mechanism for single- and multiple-part
1189 encryption and decryption with SKIPJACK in 64-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 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

1195 2.8.9 SKIPJACK-CFB32

1196 SKIPJACK-CFB32, denoted **CKM_SKIPJACK_CFB32**, is a mechanism for single- and multiple-part
1197 encryption and decryption with SKIPJACK in 32-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 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

1203 2.8.10 SKIPJACK-CFB16

1204 SKIPJACK-CFB16, denoted **CKM_SKIPJACK_CFB16**, is a mechanism for single- and multiple-part
1205 encryption and decryption with SKIPJACK in 16-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 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
-----------	----------	---------------	----------------------	---------------

1211 2.8.11 SKIPJACK-CFB8

1212 SKIPJACK-CFB8, denoted **CKM_SKIPJACK_CFB8**, is a mechanism for single- and multiple-part
1213 encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.

1214 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1215 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1216 encrypting. It MAY, of course, specify a particular IV when decrypting.

1217 Constraints on key types and the length of data are summarized in the following table:

1218 *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

1219 2.8.12 SKIPJACK-WRAP

1220 The SKIPJACK-WRAP mechanism, denoted **CKM_SKIPJACK_WRAP**, is used to wrap and unwrap a
1221 secret key (MEK). It MAY wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.

1222 It does not have a parameter.

1223 2.8.13 SKIPJACK-PRIVATE-WRAP

1224 The SKIPJACK-PRIVATE-WRAP mechanism, denoted **CKM_SKIPJACK_PRIVATE_WRAP**, is used to
1225 wrap and unwrap a private key. It MAY wrap KEA and DSA private keys.

1226 It has a parameter, a **CK_SKIPJACK_PRIVATE_WRAP_PARAMS** structure.

1227 2.8.14 SKIPJACK-RELAYX

1228 The SKIPJACK-RELAYX mechanism, denoted **CKM_SKIPJACK_RELAYX**, is used with the **C_WrapKey**
1229 function to “change the wrapping” on a private key which was wrapped with the SKIPJACK-PRIVATE-
1230 WRAP mechanism (See Section 2.8.13).

1231 It has a parameter, a **CK_SKIPJACK_RELAYX_PARAMS** structure.

1232 Although the SKIPJACK-RELAYX mechanism is used with **C_WrapKey**, it differs from other key-
1233 wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to
1234 **C_WrapKey**; however for the SKIPJACK_RELAYX mechanism, the [always invalid] value 0 should be
1235 passed as the key handle for **C_WrapKey**, and the already-wrapped key should be passed in as part of
1236 the **CK_SKIPJACK_RELAYX_PARAMS** structure.

1237 2.9 BATON

1238 2.9.1 Definitions

1239 This section defines the key type “CKK_BATON” for type CK_KEY_TYPE as used in the
1240 CKA_KEY_TYPE attribute of key objects.

1241 Mechanisms:

1242 CKM_BATON_KEY_GEN

1243 CKM_BATON_ECB128

1244 CKM_BATON_ECB96

1245 CKM_BATON_CBC128
 1246 CKM_BATON_COUNTER
 1247 CKM_BATON_SHUFFLE
 1248 CKM_BATON_WRAP

1249 2.9.2 BATON secret key objects

1250 BATON secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_BATON**) hold single-length
 1251 BATON keys. The following table defines the BATON secret key object attributes, in addition to the
 1252 common attributes defined for this object class:

1253 *Table 39, BATON Secret Key Object*

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

1254 Refer to [PKCS #11-Base] table 15 for footnotes

1255

1256 BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or
 1257 unwrap a BATON key with incorrect checksum bits MUST return an error.

1258 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key
 1259 with a specified value. Nonetheless, we provide templates for doing so.

1260 The following is a sample template for creating a BATON MEK secret key object:

```

1261 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1262 CK_KEY_TYPE keyType = CKK_BATON;
1263 CK_UTF8CHAR label[] = "A BATON MEK secret key object";
1264 CK_BYTE value[40] = {...};
1265 CK_BBOOL true = CK_TRUE;
1266 CK_ATTRIBUTE template[] = {
1267     {CKA_CLASS, &class, sizeof(class)},
1268     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1269     {CKA_TOKEN, &>true, sizeof(true)},
1270     {CKA_LABEL, label, sizeof(label)-1},
1271     {CKA_ENCRYPT, &>true, sizeof(true)},
1272     {CKA_VALUE, value, sizeof(value)}
1273 };
  
```

1274 The following is a sample template for creating a BATON TEK secret key object:

```

1275 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1276 CK_KEY_TYPE keyType = CKK_BATON;
1277 CK_UTF8CHAR label[] = "A BATON TEK secret key object";
1278 CK_BYTE value[40] = {...};
1279 CK_BBOOL true = CK_TRUE;
1280 CK_ATTRIBUTE template[] = {
1281     {CKA_CLASS, &class, sizeof(class)},
1282     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1283     {CKA_TOKEN, &>true, sizeof(true)},
1284     {CKA_LABEL, label, sizeof(label)-1},
1285     {CKA_ENCRYPT, &>true, sizeof(true)},
1286     {CKA_WRAP, &>true, sizeof(true)},
1287     {CKA_VALUE, value, sizeof(value)}
1288 };
  
```

1289 2.9.3 BATON key generation

1290 The BATON key generation mechanism, denoted **CKM_BATON_KEY_GEN**, is a key generation
 1291 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

1292 It does not have a parameter.
 1293 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
 1294 key.

1295 **2.9.4 BATON-ECB128**

1296 BATON-ECB128, denoted **CKM_BATON_ECB128**, is a mechanism for single- and multiple-part
 1297 encryption and decryption with BATON in 128-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 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

1303 **2.9.5 BATON-ECB96**

1304 BATON-ECB96, denoted **CKM_BATON_ECB96**, is a mechanism for single- and multiple-part encryption
 1305 and decryption with BATON in 96-bit electronic codebook 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 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

1311 **2.9.6 BATON-CBC128**

1312 BATON-CBC128, denoted **CKM_BATON_CBC128**, is a mechanism for single- and multiple-part
 1313 encryption and decryption with BATON in 128-bit cipher-block chaining 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 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

1319 2.9.7 BATON-COUNTER

1320 BATON-COUNTER, denoted **CKM_BATON_COUNTER**, is a mechanism for single- and multiple-part
1321 encryption and decryption with BATON in counter 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 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

1327 2.9.8 BATON-SHUFFLE

1328 BATON-SHUFFLE, denoted **CKM_BATON_SHUFFLE**, is a mechanism for single- and multiple-part
1329 encryption and decryption with BATON in shuffle mode.

1330 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1331 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1332 encrypting. It MAY, of course, specify a particular IV when decrypting.

1333 Constraints on key types and the length of data are summarized in the following table:

1334 *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

1335 2.9.9 BATON WRAP

1336 The BATON wrap and unwrap mechanism, denoted **CKM_BATON_WRAP**, is a function used to wrap
1337 and unwrap a secret key (MEK). It MAY wrap and unwrap SKIPJACK, BATON and JUNIPER keys.

1338 It has no parameters.

1339 When used to unwrap a key, this mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and
1340 **CKA_VALUE** attributes to it.

1341 2.10 JUNIPER

1342 2.10.1 Definitions

1343 This section defines the key type “CKK_JUNIPER” for type CK_KEY_TYPE as used in the
1344 CKA_KEY_TYPE attribute of key objects.

1345 Mechanisms:

1346 CKM_JUNIPER_KEY_GEN

1347 CKM_JUNIPER_ECB128

1348 CKM_JUNIPER_CBC128

1349 CKM_JUNIPER_COUNTER

1350 CKM_JUNIPER_SHUFFLE

1351 CKM_JUNIPER_WRAP

1352 2.10.2 JUNIPER secret key objects

1353 JUNIPER secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_JUNIPER**) hold single-
1354 length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to
1355 the common attributes defined for this object class:

1356 *Table 45, JUNIPER Secret Key Object*

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

1357 Refer to [PKCS #11-Base] table 15 for footnotes

1358

1359 JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or
1360 unwrap a BATON key with incorrect checksum bits MUST return an error.

1361 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key
1362 with a specified value. Nonetheless, we provide templates for doing so.

1363 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1364 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1365 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1366 CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";  
1367 CK_BYTE value[40] = {...};  
1368 CK_BBOOL true = CK_TRUE;  
1369 CK_ATTRIBUTE template[] = {  
1370     {CKA_CLASS, &class, sizeof(class)},  
1371     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1372     {CKA_TOKEN, &true, sizeof(true)},  
1373     {CKA_LABEL, label, sizeof(label)-1},  
1374     {CKA_ENCRYPT, &true, sizeof(true)},  
1375     {CKA_VALUE, value, sizeof(value)}  
1376 };
```

1377 The following is a sample template for creating a JUNIPER TEK secret key object:

```
1378 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1379 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1380 CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";  
1381 CK_BYTE value[40] = {...};  
1382 CK_BBOOL true = CK_TRUE;  
1383 CK_ATTRIBUTE template[] = {  
1384     {CKA_CLASS, &class, sizeof(class)},  
1385     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1386     {CKA_TOKEN, &true, sizeof(true)},  
1387     {CKA_LABEL, label, sizeof(label)-1},  
1388     {CKA_ENCRYPT, &true, sizeof(true)},  
1389     {CKA_WRAP, &true, sizeof(true)},  
1390     {CKA_VALUE, value, sizeof(value)}  
1391 };
```

1392 2.10.3 JUNIPER key generation

1393 The JUNIPER key generation mechanism, denoted **CKM_JUNIPER_KEY_GEN**, is a key generation
1394 mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).

1395 It does not have a parameter.

1396 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
1397 key.

1398 2.10.4 JUNIPER-ECB128

1399 JUNIPER-ECB128, denoted **CKM_JUNIPER_ECB128**, is a mechanism for single- and multiple-part
1400 encryption and decryption with JUNIPER in 128-bit electronic codebook mode.

1401 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1402 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1403 encrypting. It MAY, of course, specify a particular IV when decrypting.

1404 Constraints on key types and the length of data are summarized in the following table. For encryption
1405 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1406 *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

1407 2.10.5 JUNIPER-CBC128

1408 JUNIPER-CBC128, denoted **CKM_JUNIPER_CBC128**, is a mechanism for single- and multiple-part
1409 encryption and decryption with JUNIPER in 128-bit cipher block chaining mode.

1410 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1411 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1412 encrypting. It MAY, of course, specify a particular IV when decrypting.

1413 Constraints on key types and the length of data are summarized in the following table. For encryption
1414 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1415 *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

1416 2.10.6 JUNIPER-COUNTER

1417 JUNIPER-COUNTER, denoted **CKM_JUNIPER_COUNTER**, is a mechanism for single- and multiple-
1418 part encryption and decryption with JUNIPER in counter mode.

1419 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1420 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1421 encrypting. It MAY, of course, specify a particular IV when decrypting.

1422 Constraints on key types and the length of data are summarized in the following table. For encryption
1423 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1424 *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

1425 2.10.7 JUNIPER-SHUFFLE

1426 JUNIPER-SHUFFLE, denoted **CKM_JUNIPER_SHUFFLE**, is a mechanism for single- and multiple-part
1427 encryption and decryption with JUNIPER in shuffle mode.

1428 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1429 value generated by the token – in other words, the application MAY NOT specify a particular IV when
1430 encrypting. It MAY, of course, specify a particular IV when decrypting.

1431 Constraints on key types and the length of data are summarized in the following table. For encryption
1432 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1433 *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

1434 2.10.8 JUNIPER WRAP

1435 The JUNIPER wrap and unwrap mechanism, denoted **CKM_JUNIPER_WRAP**, is a function used to wrap
1436 and unwrap an MEK. It MAY wrap or unwrap SKIPJACK, BATON and JUNIPER keys.

1437 It has no parameters.

1438 When used to unwrap a key, this mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and
1439 **CKA_VALUE** attributes to it.

1440 2.11 MD2

1441 2.11.1 Definitions

1442 Mechanisms:

1443 CKM_MD2

1444 CKM_MD2_HMAC

1445 CKM_MD2_HMAC_GENERAL

1446 CKM_MD2_KEY_DERIVATION

1447 2.11.2 MD2 digest

1448 The MD2 mechanism, denoted **CKM_MD2**, is a mechanism for message digesting, following the MD2
1449 message-digest algorithm defined in RFC 6149.

1450 It does not have a parameter.

1451 Constraints on the length of data are summarized in the following table:

1452 *Table 50, MD2: Data Length*

Function	Data length	Digest Length
C_Digest	Any	16

1453 2.11.3 General-length MD2-HMAC

1454 The general-length MD2-HMAC mechanism, denoted **CKM_MD2_HMAC_GENERAL**, is a mechanism for
1455 signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it
1456 uses are generic secret keys.

1457 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1458 output. This length should be in the range 0-16 (the output size of MD2 is 16 bytes). Signatures (MACs)
1459 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1460 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

1461 2.11.4 MD2-HMAC

1462 The MD2-HMAC mechanism, denoted **CKM_MD2_HMAC**, is a special case of the general-length MD2-
1463 HMAC mechanism in Section 2.11.3.

1464 It has no parameter, and produces an output of length 16.

1465 2.11.5 MD2 key derivation

1466 MD2 key derivation, denoted **CKM_MD2_KEY_DERIVATION**, is a mechanism which provides the
1467 capability of deriving a secret key by digesting the value of another secret key with MD2.

1468 The value of the base key is digested once, and the result is used to make the value of the derived secret
1469 key.

- 1470 • If no length or key type is provided in the template, then the key produced by this mechanism **MUST**
1471 be a generic secret key. Its length **MUST** be 16 bytes (the output size of MD2)..
- 1472 • If no key type is provided in the template, but a length is, then the key produced by this mechanism
1473 **MUST** be a generic secret key of the specified length.
- 1474 • If no length was provided in the template, but a key type is, then that key type must have a well-
1475 defined length. If it does, then the key produced by this mechanism **MUST** be of the type specified in
1476 the template. If it doesn't, an error **MUST** be returned.
- 1477 • If both a key type and a length are provided in the template, the length must be compatible with that
1478 key type. The key produced by this mechanism **MUST** be of the specified type and length.

1479 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key **MUST** be set
1480 properly.

1481 If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.

1482 This mechanism has the following rules about key sensitivity and extractability:

- 1483 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key **MAY**
1484 both be specified to be either **CK_TRUE** or **CK_FALSE**. If omitted, these attributes each take on
1485 some default value.
- 1486 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to **CK_FALSE**, then the derived key
1487 **MUST** as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to **CK_TRUE**, then
1488 the derived key has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value as its
1489 **CKA_SENSITIVE** attribute.
- 1490 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to **CK_FALSE**, then the
1491 derived key **MUST**, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to
1492 **CK_TRUE**, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the *opposite*
1493 value from its **CKA_EXTRACTABLE** attribute.

1494 2.12 MD5

1495 2.12.1 Definitions

1496 Mechanisms:

1497 CKM_MD5

1498 CKM_MD5_HMAC

1499 CKM_MD5_HMAC_GENERAL
1500 CKM_MD5_KEY_DERIVATION

1501 2.12.2 MD5 Digest

1502 The MD5 mechanism, denoted **CKM_MD5**, is a mechanism for message digesting, following the MD5
1503 message-digest algorithm defined in RFC 1321.

1504 It does not have a parameter.

1505 Constraints on the length of input and output data are summarized in the following table. For single-part
1506 digesting, the data and the digest MAY begin at the same location in memory.

1507 *Table 52, MD5: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1508 2.12.3 General-length MD5-HMAC

1509 The general-length MD5-HMAC mechanism, denoted **CKM_MD5_HMAC_GENERAL**, is a mechanism for
1510 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it
1511 uses are generic secret keys.

1512 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1513 output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)
1514 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1515 *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

1516 2.12.4 MD5-HMAC

1517 The MD5-HMAC mechanism, denoted **CKM_MD5_HMAC**, is a special case of the general-length MD5-
1518 HMAC mechanism in Section 2.12.3.

1519 It has no parameter, and produces an output of length 16.

1520 2.12.5 MD5 key derivation

1521 MD5 key derivation denoted **CKM_MD5_KEY_DERIVATION**, is a mechanism which provides the
1522 capability of deriving a secret key by digesting the value of another secret key with MD5.

1523 The value of the base key is digested once, and the result is used to make the value of derived secret
1524 key.

- 1525 • If no length or key type is provided in the template, then the key produced by this mechanism MUST
1526 be a generic secret key. Its length MUST be 16 bytes (the output size of MD5).
- 1527 • If no key type is provided in the template, but a length is, then the key produced by this mechanism
1528 MUST be a generic secret key of the specified length.
- 1529 • If no length was provided in the template, but a key type is, then that key type must have a well-
1530 defined length. If it does, then the key produced by this mechanism MUST be of the type specified in
1531 the template. If it doesn't, an error MUST be returned.
- 1532 • If both a key type and a length are provided in the template, the length must be compatible with that
1533 key type. The key produced by this mechanism MUST be of the specified type and length.

1534 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set
 1535 properly.

1536 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.

1537 This mechanism has the following rules about key sensitivity and extractability.

- 1538 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key MAY
 1539 both be specified to either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some
 1540 default value.
- 1541 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, then the derived key
 1542 MUST as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_TRUE, then
 1543 the derived key has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value as its
 1544 **CKA_SENSITIVE** attribute.
- 1545 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE, then the
 1546 derived key MUST, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to
 1547 CK_TRUE, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the *opposite*
 1548 value from its **CKA_EXTRACTABLE** attribute.

1549 2.13 FASTHASH

1550 2.13.1 Definitions

1551 Mechanisms:
 1552 CKM_FASTHASH

1553 2.13.2 FASTHASH digest

1554 The FASTHASH mechanism, denoted **CKM_FASTHASH**, is a mechanism for message digesting,
 1555 following the U.S. government's algorithm.

1556 It does not have a parameter.

1557 Constraints on the length of input and output data are summarized in the following table:

1558 *Table 54, FASTHASH: Data Length*

Function	Input length	Digest length
C_Digest	Any	40

1559 2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)

1560 2.14.1 Definitions

1561 The mechanisms in this section are for generating keys and IVs for performing password-based
 1562 encryption. The method used to generate keys and IVs is specified in PKCS #5.

1563 Mechanisms:

- 1564 CKM_PBE_MD2_DES_CBC
- 1565 CKM_PBE_MD5_DES_CBC
- 1566 CKM_PBE_MD5_CAST_CBC
- 1567 CKM_PBE_MD5_CAST3_CBC
- 1568 CKM_PBE_MD5_CAST5_CBC
- 1569 CKM_PBE_MD5_CAST128_CBC
- 1570 CKM_PBE_SHA1_CAST5_CBC
- 1571 CKM_PBE_SHA1_CAST128_CBC

1572 CKM_PBE_SHA1_RC4_128
1573 CKM_PBE_SHA1_RC4_40
1574 CKM_PBE_SHA1_RC2_128_CBC
1575 CKM_PBE_SHA1_RC2_40_CBC

1576 2.14.2 Password-based encryption/authentication mechanism parameters

1577 2.14.2.1 CK_PBE_PARAMS; CK_PBE_PARAMS_PTR

1578 **CK_PBE_PARAMS** is a structure which provides all of the necessary information required by the
1579 CKM_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation
1580 mechanisms) and the CKM_PBA_SHA1_WITH_SHA1_HMAC mechanism. It is defined as follows:

```
1581 typedef struct CK_PBE_PARAMS {  
1582     CK_BYTE_PTR pInitVector;  
1583     CK_UTF8CHAR_PTR pPassword;  
1584     CK_ULONG ulPasswordLen;  
1585     CK_BYTE_PTR pSalt;  
1586     CK_ULONG ulSaltLen;  
1587     CK_ULONG ulIteration;  
1588 } CK_PBE_PARAMS;
```

1589 The fields of the structure have the following meanings:

1590	<i>pInitVector</i>	pointer to the location that receives the 8-byte initialization vector (IV), if an IV is required
1591		
1592	<i>pPassword</i>	points to the password to be used in the PBE key generation
1593		
1594	<i>ulPasswordLen</i>	length in bytes of the password information
1595	<i>pSalt</i>	points to the salt to be used in the PBE key generation
1596	<i>ulSaltLen</i>	length in bytes of the salt information
1597	<i>ulliteration</i>	number of iterations required for the generation

1598 **CK_PBE_PARAMS_PTR** is a pointer to a **CK_PBE_PARAMS**.

1599 2.14.3 MD2-PBE for DES-CBC

1600 MD2-PBE for DES-CBC, denoted **CKM_PBE_MD2_DES_CBC**, is a mechanism used for generating a
1601 DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an
1602 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1603 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1604 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1605 generated by the mechanism.

1606 2.14.4 MD5-PBE for DES-CBC

1607 MD5-PBE for DES-CBC, denoted **CKM_PBE_MD5_DES_CBC**, is a mechanism used for generating a
1608 DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
1609 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1610 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1611 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1612 generated by the mechanism.

1613 **2.14.5 MD5-PBE for CAST-CBC**

1614 MD5-PBE for CAST-CBC, denoted **CKM_PBE_MD5_CAST_CBC**, is a mechanism used for generating a
1615 CAST secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
1616 iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1617 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1618 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1619 generated by the mechanism

1620 The length of the CAST key generated by this mechanism MAY be specified in the supplied template; if it
1621 is not present in the template, it defaults to 8 bytes.

1622 **2.14.6 MD5-PBE for CAST3-CBC**

1623 MD5-PBE for CAST3-CBC, denoted **CKM_PBE_MD5_CAST3_CBC**, is a mechanism used for generating
1624 a CAST3 secret key and an IV from a password and a salt value by using the MD5 digest algorithm and
1625 an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1626 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1627 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1628 generated by the mechanism

1629 The length of the CAST3 key generated by this mechanism MAY be specified in the supplied template; if
1630 it is not present in the template, it defaults to 8 bytes.

1631 **2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC)**

1632 MD5-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM_PBE_MD5_CAST128_CBC** or
1633 **CKM_PBE_MD5_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
1634 and an IV from a password and a salt value by using the MD5 digest algorithm and an iteration count.
1635 This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1636 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1637 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1638 generated by the mechanism

1639 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied
1640 template; if it is not present in the template, it defaults to 8 bytes.

1641 **2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)**

1642 SHA-1-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM_PBE_SHA1_CAST128_CBC** or
1643 **CKM_PBE_SHA1_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
1644 and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This
1645 functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1646 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1647 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
1648 generated by the mechanism

1649 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied
1650 template; if it is not present in the template, it defaults to 8 bytes

1651 **2.15 PKCS #12 password-based encryption/authentication**
1652 **mechanisms**

1653 **2.15.1 Definitions**

1654 The mechanisms in this section are for generating keys and IVs for performing password-based
1655 encryption or authentication. The method used to generate keys and IVs is based on a method that was
1656 specified in PKCS #12.

1657 We specify here a general method for producing various types of pseudo-random bits from a password,
1658 p ; a string of salt bits, s ; and an iteration count, c . The “type” of pseudo-random bits to be produced is
1659 identified by an identification byte, ID , described at the end of this section.

1660 Let H be a hash function built around a compression function $f: \mathbf{Z}_2^u \times \mathbf{Z}_2^v \rightarrow \mathbf{Z}_2^u$ (that is, H has a chaining
1661 variable and output of length u bits, and the message input to the compression function of H is v bits). For
1662 MD2 and MD5, $u=128$ and $v=512$; for SHA-1, $u=160$ and $v=512$.

1663 We assume here that u and v are both multiples of 8, as are the lengths in bits of the password and salt
1664 strings and the number n of pseudo-random bits required. In addition, u and v are of course nonzero.

- 1665 1. Construct a string, D (the “diversifier”), by concatenating $v/8$ copies of ID .
- 1666 2. Concatenate copies of the salt together to create a string S of length $v \cdot \lceil s/v \rceil$ bits (the final copy of
1667 the salt MAY be truncated to create S). Note that if the salt is the empty string, then so is S .
- 1668 3. Concatenate copies of the password together to create a string P of length $v \cdot \lceil p/v \rceil$ bits (the final
1669 copy of the password MAY be truncated to create P). Note that if the password is the empty
1670 string, then so is P .
- 1671 4. Set $I=S||P$ to be the concatenation of S and P .
- 1672 5. Set $j=\lceil n/u \rceil$.
- 1673 6. For $i=1, 2, \dots, j$, do the following:
 - 1674 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
1675 of that hash; etc.; continue in this fashion until a total of c hashes have been computed,
1676 each on the result of the previous hash.
 - 1677 b. Concatenate copies of A_i to create a string B of length v bits (the final copy of A_i MAY be
1678 truncated to create B).
 - 1679 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
1680 by setting $I_j=(I_j+B+1) \bmod 2^v$ for each j . To perform this addition, treat each v -bit block as
1681 a binary number represented most-significant bit first.
- 1682 7. Concatenate A_1, A_2, \dots, A_j together to form a pseudo-random bit string, A .
- 1683 8. Use the first n bits of A as the output of this entire process

1684 When the password-based encryption mechanisms presented in this section are used to generate a key
1685 and IV (if needed) from a password, salt, and an iteration count, the above algorithm is used. To
1686 generate a key, the identifier byte ID is set to the value 1; to generate an IV, the identifier byte ID is set to
1687 the value 2.

1688 When the password-based authentication mechanism presented in this section is used to generate a key
1689 from a password, salt and an iteration count, the above algorithm is used. The identifier ID is set to the
1690 value 3.

1691 **2.15.2 SHA-1-PBE for 128-bit RC4**

1692 SHA-1-PBE for 128-bit RC4, denoted **CKM_PBE_SHA1_RC4_128**, is a mechanism used for generating
1693 a 128-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
1694 iteration count. The method used to generate the key is described above.

1695 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1696 key generation process. The parameter also has a field to hold the location of an application-supplied

1697 buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not
1698 require an IV.
1699 The key produced by this mechanism will typically be used for performing password-based encryption.

1700 **2.15.3 SHA-1_PBE for 40-bit RC4**

1701 SHA-1-PBE for 40-bit RC4, denoted **CKM_PBE_SHA1_RC4_40**, is a mechanism used for generating a
1702 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
1703 iteration count. The method used to generate the key is described above.
1704 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1705 key generation process. The parameter also has a field to hold the location of an application-supplied
1706 buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not
1707 require an IV.
1708 The key produced by this mechanism will typically be used for performing password-based encryption.

1709 **2.15.4 SHA-1_PBE for 128-bit RC2-CBC**

1710 SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_128_CBC**, is a mechanism used for
1711 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest
1712 algorithm and an iteration count. The method used to generate the key and IV is described above.
1713 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1714 key generation process and the location of an application-supplied buffer which receives the 8-byte IV
1715 generated by the mechanism.
1716 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
1717 of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object
1718 Identifier `pbeWithSHA1And128BitRC2-CBC`.
1719 The key and IV produced by this mechanism will typically be used for performing password-based
1720 encryption.

1721 **2.15.5 SHA-1_PBE for 40-bit RC2-CBC**

1722 SHA-1-PBE for 40-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_40_CBC**, is a mechanism used for
1723 generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm
1724 and an iteration count. The method used to generate the key and IV is described above.
1725 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1726 key generation process and the location of an application-supplied buffer which receives the 8-byte IV
1727 generated by the mechanism.
1728 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
1729 of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object
1730 Identifier `pbeWithSHA1And40BitRC2-CBC`.
1731 The key and IV produced by this mechanism will typically be used for performing password-based
1732 encryption.

1733 **2.16 RIPE-MD**

1734 **2.16.1 Definitions**

1735 Mechanisms:

- 1736 `CKM_RIPEMD128`
- 1737 `CKM_RIPEMD128_HMAC`
- 1738 `CKM_RIPEMD128_HMAC_GENERAL`
- 1739 `CKM_RIPEMD160`

1740 CKM_RIPEMD160_HMAC
1741 CKM_RIPEMD160_HMAC_GENERAL

1742 2.16.2 RIPE-MD 128 Digest

1743 The RIPE-MD 128 mechanism, denoted **CKM_RIPEMD128**, is a mechanism for message digesting,
1744 following the RIPE-MD 128 message-digest algorithm.

1745 It does not have a parameter.

1746 Constraints on the length of data are summarized in the following table:

1747 *Table 55, RIPE-MD 128: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1748

1749 2.16.3 General-length RIPE-MD 128-HMAC

1750 The general-length RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC_GENERAL**, is
1751 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 128
1752 hash function. The keys it uses are generic secret keys.

1753 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1754 output. This length should be in the range 0-16 (the output size of RIPE-MD 128 is 16 bytes). Signatures
1755 (MACs) produced by this mechanism **MUST** be taken from the start of the full 16-byte HMAC output.

1756 *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

1757 2.16.4 RIPE-MD 128-HMAC

1758 The RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC**, is a special case of the
1759 general-length RIPE-MD 128-HMAC mechanism in Section 2.16.3.

1760 It has no parameter, and produces an output of length 16.

1761 2.16.5 RIPE-MD 160

1762 The RIPE-MD 160 mechanism, denoted **CKM_RIPEMD160**, is a mechanism for message digesting,
1763 following the RIPE-MD 160 message-digest defined in ISO-10118.

1764 It does not have a parameter.

1765 Constraints on the length of data are summarized in the following table:

1766 *Table 57, RIPE-MD 160: Data Length*

Function	Data length	Digest length
C_Digest	Any	20

1767 **2.16.6 General-length RIPE-MD 160-HMAC**

1768 The general-length RIPE-MD 160-HMAC mechanism, denoted **CKM_RIPEMD160_HMAC_GENERAL**, is
1769 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160
1770 hash function. The keys it uses are generic secret keys.

1771 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1772 output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures
1773 (MACs) produced by this mechanism **MUST** be taken from the start of the full 20-byte HMAC output.

1774 *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

1775 **2.16.7 RIPE-MD 160-HMAC**

1776 The RIPE-MD 160-HMAC mechanism, denoted **CKM_RIPEMD160_HMAC**, is a special case of the
1777 general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.

1778 It has no parameter, and produces an output of length 20.

1779 **2.17 SET**

1780 **2.17.1 Definitions**

1781 Mechanisms:

1782 **CKM_KEY_WRAP_SET_OAEP**

1783 **2.17.2 SET mechanism parameters**

1784 **2.17.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS;**
1785 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR**

1786 **CK_KEY_WRAP_SET_OAEP_PARAMS** is a structure that provides the parameters to the
1787 **CKM_KEY_WRAP_SET_OAEP** mechanism. It is defined as follows:

```
1788 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {  
1789     CK_BYTE bBC;  
1790     CK_BYTE_PTR pX;  
1791     CK_ULONG ulXLen;  
1792 } CK_KEY_WRAP_SET_OAEP_PARAMS;
```

1793 The fields of the structure have the following meanings:

1794 *bBC* block contents byte

1795 *pX* concatenation of hash of plaintext data (if present) and
1796 extra data (if present)

1797 *ulXLen* length in bytes of concatenation of hash of plaintext data
1798 (if present) and extra data (if present). 0 if neither is
1799 present.

1800 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR** is a pointer to a
1801 **CK_KEY_WRAP_SET_OAEP_PARAMS**.

1802 2.17.3 OAEP key wrapping for SET

1803 The OAEP key wrapping for SET mechanism, denoted **CKM_KEY_WRAP_SET_OAEP**, is a mechanism
1804 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some
1805 extra data MAY be wrapped together with the DES key. This mechanism is defined in the SET protocol
1806 specifications.

1807 It takes a parameter, a **CK_KEY_WRAP_SET_OAEP_PARAMS** structure. This structure holds the
1808 "Block Contents" byte of the data and the concatenation of the hash of plaintext data (if present) and the
1809 extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by
1810 the *ulXLen* field having the value 0.

1811 When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present)
1812 and the extra data (if present) is returned following the convention described [PKCS #11-Curr],
1813 **Miscellaneous simple key derivation mechanisms**. Note that if the inputs to **C_UnwrapKey** are such
1814 that the extra data is not returned (e.g. the buffer supplied in the
1815 **CK_KEY_WRAP_SET_OAEP_PARAMS** structure is **NULL_PTR**), then the unwrapped key object MUST
1816 NOT be created, either.

1817 Be aware that when this mechanism is used to unwrap a key, the *bBC* and *pX* fields of the parameter
1818 supplied to the mechanism MAY be modified.

1819 If an application uses **C_UnwrapKey** with **CKM_KEY_WRAP_SET_OAEP**, it may be preferable for it
1820 simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data
1821 (this concatenation MUST NOT be larger than 128 bytes), rather than calling **C_UnwrapKey** twice. Each
1822 call of **C_UnwrapKey** with **CKM_KEY_WRAP_SET_OAEP** requires an RSA decryption operation to be
1823 performed, and this computational overhead MAY be avoided by this means.

1824 2.18 LYNKS

1825 2.18.1 Definitions

1826 Mechanisms:

1827 **CKM_KEY_WRAP_LYNKS**

1828 2.18.2 LYNKS key wrapping

1829 The LYNKS key wrapping mechanism, denoted **CKM_KEY_WRAP_LYNKS**, is a mechanism for
1830 wrapping and unwrapping secret keys with DES keys. It MAY wrap any 8-byte secret key, and it produces
1831 a 10-byte wrapped key, containing a cryptographic checksum.

1832 It does not have a parameter.

1833 To wrap an 8-byte secret key *K* with a DES key *W*, this mechanism performs the following steps:

- 1834 1. Initialize two 16-bit integers, sum_1 and sum_2 , to 0
- 1835 2. Loop through the bytes of *K* from first to last.
- 1836 3. Set $sum_1 = sum_1 + \text{the key byte}$ (treat the key byte as a number in the range 0-255).
- 1837 4. Set $sum_2 = sum_2 + sum_1$.
- 1838 5. Encrypt *K* with *W* in ECB mode, obtaining an encrypted key, *E*.
- 1839 6. Concatenate the last 6 bytes of *E* with sum_2 , representing sum_2 most-significant bit first. The
1840 result is an 8-byte block, *T*
- 1841 7. Encrypt *T* with *W* in ECB mode, obtaining an encrypted checksum, *C*.
- 1842 8. Concatenate *E* with the last 2 bytes of *C* to obtain the wrapped key.

1843 When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly,
1844 an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity
1845 bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.

1846

1847 **3 PKCS #11 Implementation Conformance**

1848 An implementation is a conforming implementation if it meets the conditions specified in one or more
1849 server profiles specified in **[PKCS #11-Prof]**.

1850 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1851 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL
1852 conform to all normative statements within the clauses specified for that profile and for any subclauses to
1853 each of those clauses .

1854

1855 Appendix A. Acknowledgments

1856 The following individuals have participated in the creation of this specification and are gratefully
1857 acknowledged:

1858

1859 **Participants:**

1860 Gil Abel, Athena Smartcard Solutions, Inc.

1861 Warren Armstrong, QuintessenceLabs

1862 Peter Bartok, Venafi, Inc.

1863 Anthony Berglas, Cryptsoft

1864 Kelley Burgin, National Security Agency

1865 Robert Burns, Thales e-Security

1866 Wan-Teh Chang, Google Inc.

1867 Hai-May Chao, Oracle

1868 Janice Cheng, Vormetric, Inc.

1869 Sangrae Cho, Electronics and Telecommunications Research Institute (ETRI)

1870 Doron Cohen, SafeNet, Inc.

1871 Fadi Cotran, Futurex

1872 Tony Cox, Cryptsoft

1873 Christopher Duane, EMC

1874 Chris Dunn, SafeNet, Inc.

1875 Valerie Fenwick, Oracle

1876 Terry Fletcher, SafeNet, Inc.

1877 Susan Gleeson, Oracle

1878 Sven Gossel, Charismathics

1879 Robert Griffin, EMC

1880 Paul Grojean, Individual

1881 Peter Gutmann, Individual

1882 Dennis E. Hamilton, Individual

1883 Thomas Hardjono, M.I.T.

1884 Tim Hudson, Cryptsoft

1885 Gershon Janssen, Individual

1886 Seunghun Jin, Electronics and Telecommunications Research Institute (ETRI)

1887 Andrey Jivsov, Symantec Corp.

1888 Greg Kazmierczak, Wave Systems Corp.

1889 Mark Knight, Thales e-Security

1890 Darren Krahn, Google Inc.

1891 Alex Krasnov, Infineon Technologies AG

1892 Dina Kurktchi-Nimeh, Oracle

1893 Mark Lambiase, SecureAuth Corporation

1894 Lawrence Lee, GoTrust Technology Inc.

1895 John Leiseboer, QuintessenceLabs
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 Johann Schoetz, Infineon Technologies AG
1909 Rayees Shamsuddin, Wave Systems Corp.
1910 Radhika Siravara, Oracle
1911 Brian Smith, Mozilla Corporation
1912 David Smith, Venafi, Inc.
1913 Ryan Smith, Futurex
1914 Jerry Smith, US Department of Defense (DoD)
1915 Oscar So, Oracle
1916 Michael Stevens, QuintessenceLabs
1917 Michael StJohns, Individual
1918 Sander Temme, Thales e-Security
1919 Kiran Thota, VMware, Inc.
1920 Walter-John Turnes, Gemini Security Solutions, Inc.
1921 Stef Walter, Red Hat
1922 Jeff Webb, Dell
1923 Magda Zdunkiewicz, Cryptsoft
1924 Chris Zimman, Bloomberg Finance L.P.

1925

Appendix B. Manifest constants

1926 The following constants have been defined for PKCS #11 V2.40. Also, refer to **[PKCS #11-Base]** and
1927 **[PKCS #11-Curr]** for additional definitions.

```
1928 /*  
1929 * Copyright OASIS Open 2014. All rights reserved.  
1930 * OASIS trademark, IPR and other policies apply.  
1931 * http://www.oasis-open.org/policies-guidelines/ipr  
1932 */  
1933  
1934 #define CKK_KEA 0x00000005  
1935 #define CKK_RC2 0x00000011  
1936 #define CKK_RC4 0x00000012  
1937 #define CKK_DES 0x00000013  
1938 #define CKK_CAST 0x00000016  
1939 #define CKK_CAST3 0x00000017  
1940 #define CKK_CAST5 0x00000018  
1941 #define CKK_CAST128 0x00000018  
1942 #define CKK_RC5 0x00000019  
1943 #define CKK_IDEA 0x0000001A  
1944 #define CKK_SKIPJACK 0x0000001B  
1945 #define CKK_BATON 0x0000001C  
1946 #define CKK_JUNIPER 0x0000001D  
1947 #define CKM_MD2_RSA_PKCS 0x00000004  
1948 #define CKM_MD5_RSA_PKCS 0x00000005  
1949 #define CKM_RIPEMD128_RSA_PKCS 0x00000007  
1950 #define CKM_RIPEMD160_RSA_PKCS 0x00000008  
1951 #define CKM_RC2_KEY_GEN 0x00000100  
1952 #define CKM_RC2_ECB 0x00000101  
1953 #define CKM_RC2_CBC 0x00000102  
1954 #define CKM_RC2_MAC 0x00000103  
1955 #define CKM_RC2_MAC_GENERAL 0x00000104  
1956 #define CKM_RC2_CBC_PAD 0x00000105  
1957 #define CKM_RC4_KEY_GEN 0x00000110  
1958 #define CKM_RC4 0x00000111  
1959 #define CKM_DES_KEY_GEN 0x00000120  
1960 #define CKM_DES_ECB 0x00000121  
1961 #define CKM_DES_CBC 0x00000122  
1962 #define CKM_DES_MAC 0x00000123  
1963 #define CKM_DES_MAC_GENERAL 0x00000124  
1964 #define CKM_DES_CBC_PAD 0x00000125  
1965 #define CKM_MD2 0x00000200  
1966 #define CKM_MD2_HMAC 0x00000201  
1967 #define CKM_MD2_HMAC_GENERAL 0x00000202  
1968 #define CKM_MD5 0x00000210  
1969 #define CKM_MD5_HMAC 0x00000211  
1970 #define CKM_MD5_HMAC_GENERAL 0x00000212  
1971 #define CKM_RIPEMD128 0x00000230  
1972 #define CKM_RIPEMD128_HMAC 0x00000231  
1973 #define CKM_RIPEMD128_HMAC_GENERAL 0x00000232  
1974 #define CKM_RIPEMD160 0x00000240  
1975 #define CKM_RIPEMD160_HMAC 0x00000241  
1976 #define CKM_RIPEMD160_HMAC_GENERAL 0x00000242  
1977 #define CKM_CAST_KEY_GEN 0x00000300  
1978 #define CKM_CAST_ECB 0x00000301  
1979 #define CKM_CAST_CBC 0x00000302  
1980 #define CKM_CAST_MAC 0x00000303  
1981 #define CKM_CAST_MAC_GENERAL 0x00000304  
1982 #define CKM_CAST_CBC_PAD 0x00000305  
1983 #define CKM_CAST3_KEY_GEN 0x00000310
```

```

1984 #define CKM_CAST3_ECB 0x00000311
1985 #define CKM_CAST3_CBC 0x00000312
1986 #define CKM_CAST3_MAC 0x00000313
1987 #define CKM_CAST3_MAC_GENERAL 0x00000314
1988 #define CKM_CAST3_CBC_PAD 0x00000315
1989 #define CKM_CAST5_KEY_GEN 0x00000320
1990 #define CKM_CAST128_KEY_GEN 0x00000320
1991 #define CKM_CAST5_ECB 0x00000321
1992 #define CKM_CAST128_ECB 0x00000321
1993 #define CKM_CAST5_CBC 0x00000322
1994 #define CKM_CAST128_CBC 0x00000322
1995 #define CKM_CAST5_MAC 0x00000323
1996 #define CKM_CAST128_MAC 0x00000323
1997 #define CKM_CAST5_MAC_GENERAL 0x00000324
1998 #define CKM_CAST128_MAC_GENERAL 0x00000324
1999 #define CKM_CAST5_CBC_PAD 0x00000325
2000 #define CKM_CAST128_CBC_PAD 0x00000325
2001 #define CKM_RC5_KEY_GEN 0x00000330
2002 #define CKM_RC5_ECB 0x00000331
2003 #define CKM_RC5_CBC 0x00000332
2004 #define CKM_RC5_MAC 0x00000333
2005 #define CKM_RC5_MAC_GENERAL 0x00000334
2006 #define CKM_RC5_CBC_PAD 0x00000335
2007 #define CKM_IDEA_KEY_GEN 0x00000340
2008 #define CKM_IDEA_ECB 0x00000341
2009 #define CKM_IDEA_CBC 0x00000342
2010 #define CKM_IDEA_MAC 0x00000343
2011 #define CKM_IDEA_MAC_GENERAL 0x00000344
2012 #define CKM_IDEA_CBC_PAD 0x00000345
2013 #define CKM_MD5_KEY_DERIVATION 0x00000390
2014 #define CKM_MD2_KEY_DERIVATION 0x00000391
2015 #define CKM_PBE_MD2_DES_CBC 0x000003A0
2016 #define CKM_PBE_MD5_DES_CBC 0x000003A1
2017 #define CKM_PBE_MD5_CAST_CBC 0x000003A2
2018 #define CKM_PBE_MD5_CAST3_CBC 0x000003A3
2019 #define CKM_PBE_MD5_CAST5_CBC 0x000003A4
2020 #define CKM_PBE_MD5_CAST128_CBC 0x000003A4
2021 #define CKM_PBE_SHA1_CAST5_CBC 0x000003A5
2022 #define CKM_PBE_SHA1_CAST128_CBC 0x000003A5
2023 #define CKM_PBE_SHA1_RC4_128 0x000003A6
2024 #define CKM_PBE_SHA1_RC4_40 0x000003A7
2025 #define CKM_PBE_SHA1_RC2_128_CBC 0x000003AA
2026 #define CKM_PBE_SHA1_RC2_40_CBC 0x000003AB
2027 #define CKM_KEY_WRAP_LYNKS 0x00000400
2028 #define CKM_KEY_WRAP_SET_OAEP 0x00000401
2029 #define CKM_SKIPJACK_KEY_GEN 0x00001000
2030 #define CKM_SKIPJACK_ECB64 0x00001001
2031 #define CKM_SKIPJACK_CBC64 0x00001002
2032 #define CKM_SKIPJACK_OFB64 0x00001003
2033 #define CKM_SKIPJACK_CFB64 0x00001004
2034 #define CKM_SKIPJACK_CFB32 0x00001005
2035 #define CKM_SKIPJACK_CFB16 0x00001006
2036 #define CKM_SKIPJACK_CFB8 0x00001007
2037 #define CKM_SKIPJACK_WRAP 0x00001008
2038 #define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009
2039 #define CKM_SKIPJACK_RELAYX 0x0000100a
2040 #define CKM_KEA_KEY_PAIR_GEN 0x00001010
2041 #define CKM_KEA_KEY_DERIVE 0x00001011
2042 #define CKM_FORTEZZA_TIMESTAMP 0x00001020
2043 #define CKM_BATON_KEY_GEN 0x00001030
2044 #define CKM_BATON_ECB128 0x00001031
2045 #define CKM_BATON_ECB96 0x00001032
2046 #define CKM_BATON_CBC128 0x00001033
2047 #define CKM_BATON_COUNTER 0x00001034

```

```
2048 #define CKM_BATON_SHUFFLE 0x00001035
2049 #define CKM_BATON_WRAP 0x00001036
2050 #define CKM_JUNIPER_KEY_GEN 0x00001060
2051 #define CKM_JUNIPER_ECB128 0x00001061
2052 #define CKM_JUNIPER_CBC128 0x00001062
2053 #define CKM_JUNIPER_COUNTER 0x00001063
2054 #define CKM_JUNIPER_SHUFFLE 0x00001064
2055 #define CKM_JUNIPER_WRAP 0x00001065
2056 #define CKM_FASTHASH 0x00001070
```

2057

2058

Appendix C. Revision History

2059

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
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.

2060

2061