

PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40

Committee Specification Draft 01 / Public Review Draft 01

30 October 2013

Specification URIs

This 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

Previous version:

N/A

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. 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.
 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. 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. 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 http://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 (http://www.oasisopen.org/committees/pkcs11/ipr.php).

Citation format:

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

[PKCS11-hist]

PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40. 30 October 2013. OASIS Committee Specification Draft 01 / Public Review Draft 01. http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.html.

Notices

Copyright © OASIS Open 2013. 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 http://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1	Introduction	8
	1.1 Terminology	8
	1.2 Definitions	8
	1.3 Normative References	ç
	1.4 Non-Normative References	g
2	Mechanisms	13
	2.1 FORTEZZA timestamp	16
	2.2 KEA	16
	2.2.1 Definitions	16
	2.2.2 KEA mechanism parameters	16
	2.2.3 KEA public key objects	17
	2.2.4 KEA private key objects	18
	2.2.5 KEA key pair generation	18
	2.2.6 KEA key derivation	19
	2.3 RC2	20
	2.3.1 Definitions	20
	2.3.2 RC2 secret key objects	20
	2.3.3 RC2 mechanism parameters	21
	2.3.4 RC2 key generation	22
	2.3.5 RC2-ECB	22
	2.3.6 RC2-CBC	23
	2.3.7 RC2-CBC with PKCS padding	23
	2.3.8 General-length RC2-MAC	24
	2.3.9 RC2-MAC	24
	2.4 RC4	25
	2.4.1 Definitions	25
	2.4.2 RC4 secret key objects	25
	2.4.3 RC4 key generation	25
	2.4.4 RC4 mechanism	26
	2.5 RC5	26
	2.5.1 Definitions	
	2.5.2 RC5 secret key objects	26
	2.5.3 RC5 mechanism parameters	27
	2.5.4 RC5 key generation	28
	2.5.5 RC5-ECB	28
	2.5.6 RC5-CBC	
	2.5.7 RC5-CBC with PKCS padding	29
	2.5.8 General-length RC5-MAC	30
	2.5.9 RC5-MAC	30
	2.6 General block cipher	31
	2.6.1 Definitions	31
	2.6.2 DES secret key objects	32
	2.6.3 CAST secret key objects	33

2.6.4 CAST3 secret key objects	33
2.6.5 CAST128 (CAST5) secret key objects	34
2.6.6 IDEA secret key objects	34
2.6.7 CDMF secret key objects	35
2.6.8 General block cipher mechanism parameters	35
2.6.9 General block cipher key generation	35
2.6.10 General block cipher ECB	36
2.6.11 General block cipher CBC	36
2.6.12 General block cipher CBC with PCKS padding	37
2.6.13 General-length general block cipher MAC	38
2.6.14 General block cipher MAC	38
2.7 SKIPJACK	39
2.7.1 Definitions	
2.7.2 SKIPJACK secret key objects	39
2.7.3 SKIPJACK Mechanism parameters	40
2.7.4 SKIPJACK key generation	42
2.7.5 SKIPJACK-ECB64	42
2.7.6 SKIPJACK-CBC64	42
2.7.7 SKIPJACK-OFB64	42
2.7.8 SKIPJACK-CFB64	43
2.7.9 SKIPJACK-CFB32	43
2.7.10 SKIPJACK-CFB16	43
2.7.11 SKIPJACK-CFB8	44
2.7.12 SKIPJACK-WRAP	44
2.7.13 SKIPJACK-PRIVATE-WRAP	44
2.7.14 SKIPJACK-RELAYX	
2.8 BATON	44
2.8.1 Definitions	44
2.8.2 BATON secret key objects	
2.8.3 BATON key generation	45
2.8.4 BATON-ECB128	46
2.8.5 BATON-ECB96	
2.8.6 BATON-CBC128	
2.8.7 BATON-COUNTER	
2.8.8 BATON-SHUFFLE	
2.8.9 BATON WRAP	
2.9 JUNIPER	47
2.9.1 Definitions	
2.9.2 JUNIPER secret key objects	
2.9.3 JUNIPER key generation	
2.9.4 JUNIPER-ECB128	
2.9.5 JUNIPER-CBC128	
2.9.6 JUNIPER-COUNTER	
2.9.7 JUNIPER-SHUFFLE	49
298 JUNIPER WRAP	50

2.10 MD2	50
2.10.1 Definitions	50
2.10.2 MD2 digest	50
2.10.3 General-length MD2-HMAC	50
2.10.4 MD2-HMAC	51
2.10.5 MD2 key derivation	51
2.11 MD5	51
2.11.1 Definitions	51
2.11.2 MD5 Digest	52
2.11.3 General-length MD5-HMAC	52
2.11.4 MD5-HMAC	52
2.11.5 MD5 key derivation	52
2.12 FASTHASH	53
2.12.1 Definitions	53
2.12.2 FASTHASH digest	53
2.13 PKCS #5 and PKCS #5-style password-based encryption (PBD)	53
2.13.1 Definitions	53
2.13.2 Password-based encryption/authentication mechanism parameters	54
2.13.3 MD2-PBE for DES-CBC	54
2.13.4 MD5-PBE for DES-CBC	54
2.13.5 MD5-PBE for CAST-CBC	55
2.13.6 MD5-PBE for CAST3-CBC	55
2.13.7 MD5-PBE for CAST128-CBC (CAST5-CBC)	55
2.13.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)	55
2.14 PKCS #12 password-based encryption/authentication mechanisms	56
2.14.1 SHA-1-PBE for 128-bit RC4	56
2.14.2 SHA-1_PBE for 40-bit RC4	57
2.14.3 SHA-1_PBE for 128-bit RC2-CBC	57
2.14.4 SHA-1_PBE for 40-bit RC2-CBC	57
2.15 RIPE-MD	57
2.15.1 Definitions	57
2.15.2 RIPE-MD 128 Digest	58
2.15.3 General-length RIPE-MD 128-HMAC	58
2.15.4 RIPE-MD 128-HMAC	58
2.15.5 RIPE-MD 160	58
2.15.6 General-length RIPE-MD 160-HMAC	58
2.15.7 RIPE-MD 160-HMAC	59
2.16 SET	59
2.16.1 Definitions	59
2.16.2 SET mechanism parameters	59
2.16.3 OAEP key wrapping for SET	59
2.17 LYNKS	
2.17.1 Definitions	60
2.17.2 LYNKS key wrapping	60
PKCS #11 Implementation Conformance	

Appendix A.	Acknowledgments	62
• •	Manifest constants	
Appendix C.	Revision History	67

1 Introduction

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

4 5

6

1

All text is normative unless otherwise labeled.

1.1 Terminology

7	The key words "MUST", '	'MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
8		NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be
9		interpreted as described in [PKCS #11-Base] PKCS #11 Cryptographic Token
10		Interface Base Specification Version 2.40. Latest version. http://docs.oasis-
11		open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html.
12	[PKCS #11-Curr]	PKCS #11 Cryptographic Token Interface Current Mechanisms Specification
13		Version 2.40. Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-
14		curr/v2.40/pkcs11-curr-v2.40.html.
15	[PKCS #11-Prof]	PKCS #11 Cryptographic Token Interface Profiles Version 2.40. Latest version.
16		http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-
17		v2.40.html.
18	[RFC2119].	

19

20

21 22

1.2 Definitions

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

23	BATON	MISSI's BATON block cipher.
24	CAST	Entrust Technologies' proprietary symmetric block cipher
25	CAST3	Entrust Technologies' proprietary symmetric block cipher
26 27	CAST5	Another name for Entrust Technologies' symmetric block cipher CAST128. CAST128 is the preferred name.
28	CAST128	Entrust Technologies' symmetric block cipher.
29 30 31	CDMF	Commercial Data Masking Facility, a block encipherment method specified by International Business Machines Corporation and based on DES.
32	CMS	Cryptographic Message Syntax (see RFC 2630)
33	DES	Data Encryption Standard, as defined in FIPS PUB 46-3
34	ECB	Electronic Codebook mode, as defined in FIPS PUB 81.
35	FASTHASH	MISSI's FASTHASH message-digesting algorithm.
36	IDEA	Ascom Systec's symmetric block cipher.
37	IV	Initialization Vector.
38	JUNIPER	MISSI's JUNIPER block cipher.
39	KEA	MISSI's Key Exchange Algorithm.
40	LYNKS	A smart card manufactured by SPYRUS.

41		MAC	Message Authentication Code
42 43		MD2	RSA Security's MD2 message-digest algorithm, as defined in RFC 1319.
44 45		MD5	RSA Security's MD5 message-digest algorithm, as defined in RFC 1321.
46		PRF	Pseudo random function.
47		RSA	The RSA public-key cryptosystem.
48		RC2	RSA Security's RC2 symmetric block cipher.
49		RC4	RSA Security's proprietary RC4 symmetric stream cipher.
50		RC5	RSA Security's RC5 symmetric block cipher.
51		SET	The Secure Electronic Transaction protocol.
52 53		SHA-1	The (revised) Secure Hash Algorithm with a 160-bit message digest, as defined in FIPS PUB 180-2.
54	s	KIPJACK	MISSI's SKIPJACK block cipher.
55 56 57		UTF-8	Universal Character Set (UCS) transformation format (UTF) that represents ISO 10646 and UNICODE strings with a variable number of octets
58	1.3 Normative F	Reference	es
59 60 61	[PKCS #11-Base]		Cryptographic Token Interface Base Specification Version 2.40. Latest v://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-
62 63 64	[PKCS #11-Curr]	Version 2.40	Cryptographic Token Interface Current Mechanisms Specification O. Latest version. http://docs.oasis-open.org/pkcs11/pkcs11- kcs11-curr-v2.40.html.
65 66 67	[PKCS #11-Prof]	PKCS #11 (Cryptographic Token Interface Profiles Version 2.40. Latest version. asis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-
68 69	[RFC2119]	Bradner, S.	, "Key words for use in RFCs to Indicate Requirement Levels", BCP 19, March 1997. http://www.ietf.org/rfc/rfc2119.txt.
70	1.4 Non-Norma	tive Refe	rences
71	[ANSI C]		American National Standard for Programming Languages – C. 1990
72 73	[ANSI X9.31]		Standards Committee X9. Digital Signatures Using Reversible Public graphy for the Financial Services Industry (rDSA). 1998.
74	[ANSI X9.42]		Standards Committee X9. Public Key Cryptography for the Financial
75 70	-		dustry: Agreement of Symmetric Keys Using Discrete Logarithm
76 77	[ANSI X9.62]	Cryptograpi Accredited 5	ny. 2003 Standards Committee X9. <i>Public Key Cryptography for the Financial</i>
78	[Services Inc	dustry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998
79 80 81	[CC/PP]	Vocabularie	posite Capability/Preference Profiles (CC/PP): Structure and es. World Wide Web Consortium, January 2004. URL: w3.org/RT/CCPP-struct-vocab/
82	[CDPD]	Ameritech N	Mobile Communications et al. Cellular Digital Packet Data System
83 84 85	[FIPS PUB 46-3]	NIST. FIPS	ns: Part 406: Airlink Security. 1993 6 46-3: Data Encryption Standard (DES). October 26, 2999. URL: ist.gov/publications/fips/index.html

		NOT 5/20 5 4 0 4 4 4 4 4 4 4 4 4 4 4 5 5 6 7 4 4 4 4 4 5 6 7 5 6 7 4 5 6 7 4 5 6 7 6 7 6 7 6 7 6 7 6 7 6 7 6 7
86 87	[FIPS PUB 74]	NIST. FIPS 74: Guidelines for Implementing and Using the NBS Data Encryption Standard. April 1, 1981. URL: http://csrc.nist.gov/publications/fips/index.html
88	[FIPS PUB 81]	NIST. FIPS 81: DES Modes of Operation. December 1980. URL:
89		http://csrc.nist.gov/publications/fips/index.html
90 91	[FIPS PUB 113]	NIST. FIPS 113: Computer Data Authentication. May 30, 1985. URL: http://csrc.nist.gov/publications/fips/index.html
92	[FIPS PUB 180-2]	NIST. FIPS 180-2: Secure Hash Standard. August 1, 2002. URL:
93		http://csrc.nist.gov/publications/fips/index.html
94 95	[FIPS PUB 186-2]	NIST. FIPS 186-2: Digital Signature Standard. January 27, 2000. URL: http://csrc.nist.gov/publications/fips/index.html
96 97	[FIPS PUB 197]	NIST. FIPS 197: Advanced Encryption Standard (AES). November 26, 2001. URL: http://csrc.nist.gov/publications/fips/index.html
98 99	[FORTEZZA CIPG]	NSA, Workstation Security Products. FORTEZZA Cryptologic Interface Programmers Guide, Revision 1.52. November 1985
100 101	[GCS-API]	X/Open Company Ltd. Generic Cryptographic Service API (GCS-API), Base – Draft 2. February 14, 1995.
102 103	[ISO/IEC 7816-1]	ISO. Information Technology – Identification Cards – Integrated Circuit(s) with Contacts – Part 1: Physical Characteristics. 1998.
104 105	[ISO/IEC 7816-4]	ISO. Information Technology – Identification Cards – Integrated Circuit(s) with Contacts – Part 4: Interindustry Commands for Interchange. 1995.
106 107	[ISO/IEC 8824-1]	ISO. Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Base Notation. 2002.
108 109	[ISO/IEC 8825-1]	ISO. Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished
110		Encoding Rules (DER). 2002.
111 112	[ISO/IEC 9594-1]	ISO. Information Technology – Open System Interconnection – The Directory: Overview of Concepts, Models and Services. 2001.
113 114	[ISO/IEC 9594-8]	ISO. Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks. 2001.
115 116 117	[ISO/IEC 9796-2]	ISO. Information Technology – Security Techniques – Digital Signature Scheme Giving Message Recovery – Part 2: Integer factorization based mechanisms. 2002.
118 119	[Java MIDP]	Java Community Process. <i>Mobile Information Device Profile for Java 2 Micro Edition</i> . November 2002. URL: http://jcp.org/jsr/detail/118.jsp
120 121	[MeT-PTD]	MeT. MeT PTD Definition – Personal Trusted Device Definition, Version 1.0. February 2003. URL: http://www.mobiletransaction.org
122 123	[PCMCIA]	Personal Computer Memory Card International Association. <i>PC Card Standard, Release 2.1.</i> July 1993.
124	[PKCS #1]	RSA Laboratories. RSA Cryptography Standard, v2.1. June 14, 2002
125 126	[PKCS #3]	RSA Laboratories. <i>Diffie-Hellman Key-Agreement Standard, v1.4.</i> November 1993.
127 128	[PKCS #5]	RSA Laboratories. <i>Password-Based Encryption Standard, v2.0.</i> March 26, 1999.
129 130	[PKCS #7]	RSA Laboratories. <i>Cryptographic Message Syntax Standard, v1.5.</i> November 1993
131 132	[PKCS #8]	RSA Laboratories. <i>Private-Key Information Syntax Standard, v1.2.</i> November 1993.
133 134 135	[PKCS #11-UG]	PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
136 137	[PKCS #11-C]	RSA Laboratories. <i>PKCS#11: Conformance Profile Specification.</i> October 2000.
138	[PKCS #11-P]	RSA Laboratories. <i>PKCS #11 Profiles for mobile devices</i> . June 2003.
		22.2

400		
139 140	[PKCS #12]	DCA Laboratorias - Paragnal Information Evaboras Syntax Standard v1.0
140	[PRC5 #12]	RSA Laboratories. Personal Information Exchange Syntax Standard, v1.0. June 1999.
142	[RFC 1319]	B. Kaliski. <i>RFC 1319: The MD2 Message-Digest Algorithm.</i> RSA Laboratories,
143	[141 0 1010]	April 1992. URL: http://ietf.org/rfc/rfc1319.txt
144	[RFC 1321]	R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. MIT Laboratory for
145	[6 .62.]	Computer Science and RSA Data Security, Inc., April 1992. URL:
146		http://ieft.org/rfc/rfc1321.txt
147	[RFC 1421]	J. Linn. RFC 1421: Privacy Enhancement for Internet Electronic Mail: Part I:
148		Message Encryption and Authentication Procedures. IAB IRTF PSRG, IETF
149		PEM WG, February 1993. URL: http://ieft.org/rfc/rfc1421.txt
150	[RFC 2045]	Freed, N., and Borenstein. RFC 2045: Multipurpose Internet Mail Extensions
151		(MIME) Part One: Format of Internet Message Bodies. November 1996. URL:
152	[DEC 2246]	http://ieft.org/rfc/rfc2045.txt
153 154	[RFC 2246]	T. Dierks and C. Allen. <i>RFC 2245: The TLS Protocol Version 1.0.</i> Certicom, January 1999. URL: http://ieft.org/rfc/rfc2246.txt
155 156	[RFC 2279]	F. Yergeau. <i>RFC 2279: UTF-8, a transformation format of ISO 10646.</i> Alis Technologies, January 1998. URL: http://ietf.org/rfc/rfc2279.txt
157 158	[RFC 2534]	Masinter, L., Wing, D., Mutz, A., and K. Holtman. <i>RFC 2534: Media Features for Display, Print and Fax.</i> March 1999. URL: http://ieft.org/rfc/rfc2534.txt
159	[RFC 2630]	R. Houseley. <i>RFC 2630: cryptographic Message Syntax</i> . June 1999. URL:
160	[]	http://ieft.org/rfc/rfc2630.txt
161	[RFC 2743]	J. Linn. RFC 2743: Generic Security Service Application Program Interface
162		Version 2, Update 1. RSA Laboratories, January 2000. URL:
163		http://ieft.org/rfc/rfc2743.txt
164 165	[RFC 2744]	J. Wray. RFC 2744: Generic Security Services API Version 2: C-bindings. Iris Associates, January 2000. URL: http://ieft.org/rfc/rfc2744.txt
166	[SEC-1]	Standards for Efficient Cryptography Group (SECG). Standards for Efficient
167		Cryptography (SEC) 1: Elliptic Curve Cryptography. Version 1.0, September 20,
168		2000.
169	[SEC-2]	Standards for Efficient cryptography Group (SECG). Standards for Efficient
170 171		Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters. Version 1.0, September 20, 2000.
171	[TLS]	IETF. RFC 2246: The TLS Protocol Version 1.0. January 1999. URL:
172	[ILO]	http://ieft.org/rfc/rfc2256.txt
174	[WIM]	WAP. Wireless Identity Module. – WAP-260-WIP-20010712.a. July 2001. URL:
175	[]	http://www.wapforum.org
176	[WPKI]	WAP. Wireless PKI. – WAP-217-WPKI-20010424-a. April 2001. URL:
177		http://www.wapforum.org
178	[WTLS]	WAP. Wireless Transport Layer Security Version - WAP-261-WTLS-20010406-
179		a. April 2001. URL: http://www.wapforum.org
180	[X.500]	ITU-T. Information Technology – Open Systems Interconnection –The Directory:
181		Overview of Concepts, Models and Services. February 2001. (Identical to
182		ISO/IEC 9594-1)
183	[X.509]	ITU-T. Information Technology – Open Systems Interconnection – The
184 185		Directory: Public-key and Attribute Certificate Frameworks. March 2000. (Identical to ISO/IEC 9594-8)
186	LA 60UI	ITU-T. Information Technology – Abstract Syntax Notation One (ASN.1):
187	[X.680]	Specification of Basic Notation. July 2002. (Identical to ISO/IEC 8824-1)
188	[X.690]	ITU-T. Information Technology – ASN.1 Encoding Rules: Specification of Basic
189	[7,1000]	Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished
190		Encoding Rules (DER). July 2002. (Identical to ISO/IEC 8825-1)
191		

2 Mechanisms

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

195 196 197

198

199 200

201

202

203204

193

194

The following table shows which Cryptoki mechanisms are supported by different cryptographic operations. For any particular token, of course, a particular operation may well 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 can also perform RSA encryption with **CKM_RSA_PKCS**.

Table 1, Mechanisms vs. Functions

	Functions						
Mechanism	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					Х		
CKM_KEA_KEY_DERIVE							Х
CKM_RC2_KEY_GEN					Х		
CKM_RC2_ECB	Х					Х	
CKM_RC2_CBC	Х					Х	
CKM_RC2_CBC_PAD	Х					Х	
CKM_RC2_MAC_GENERAL		Х					
CKM_RC2_MAC		Х					
CKM_RC4_KEY_GEN					Х		
CKM_RC4	Х						
CKM_RC5_KEY_GEN					Х		
CKM_RC5_ECB	Х					Х	
CKM_RC5_CBC	Х					Х	
CKM_RC5_CBC_PAD	Х					Х	
CKM_RC5_MAC_GENERAL		Х					
CKM_RC5_MAC		Х					
CKM_DES_KEY_GEN					Х		
CKM_DES_ECB	Х					Х	
CKM_DES_CBC	Х					Х	
CKM_DES_CBC_PAD	Х					Х	
CKM_DES_MAC_GENERAL		Х					
CKM_DES_MAC		Х					
CKM_CAST_KEY_GEN					Х		
CKM_CAST_ECB	Х					Х	
CKM_CAST_CBC	Х					Х	
CKM_CAST_CBC_PAD	Х					Х	
CKM_CAST_MAC_GENERAL		Х					
CKM_CAST_MAC		Х					
CKM_CAST3_KEY_GEN					Х		

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_CAST3_ECB	X					Х	
CKM_CAST3_CBC	Х					Х	
CKM_CAST3_CBC_PAD	Х					Х	
CKM_CAST3_MAC_GENERAL		Х					
CKM_CAST3_MAC		Х					
CKM_CAST128_KEY_GEN					Х		
(CKM_CAST5_KEY_GEN)							
CKM_CAST128_ECB	X					Х	
(CKM_CAST5_ECB)							
CKM_CAST128_CBC	X					Х	
(CKM_CAST5_CBC)							
CKM_CAST128_CBC_PAD	X					Х	
(CKM_CAST5_CBC_PAD)							
CKM_CAST128_MAC_GENERAL		Х					
(CKM_CAST5_MAC_GENERAL) CKM_CAST128_MAC		X					
(CKM_CAST5_MAC)		^					
CKM_IDEA_KEY_GEN					Х		
CKM_IDEA_ECB	X				,	Х	
CKM_IDEA_CBC	X					X	
CKM_IDEA_CBC_PAD	X					X	
CKM_IDEA_MAC_GENERAL	, , , , , , , , , , , , , , , , , , ,	X				^	
		X					
CKM_IDEA_MAC		^			X		
CKM_CDMF_KEY_GEN					^		
CKM_CDMF_ECB	X					X	
CKM_CDMF_CBC	Х					X	
CKM_CDMF_CBC_PAD	Х					Х	
CKM_CDMF_MAC_GENERAL		Х					
CKM_CDMF_MAC		Х					
CKM_SKIPJACK_KEY_GEN					Х		
CKM_SKIPJACK_ECB64	X						
CKM_SKIPJACK_CBC64	X						
CKM_SKIPJACK_OFB64	X						
CKM_SKIPJACK_CFB64	X						
CKM_SKIPJACK_CFB32	Х						
CKM_SKIPJACK_CFB16	X						
CKM_SKIPJACK_CFB8	X						
CKM_SKIPJACK_WRAP						Х	
CKM_SKIPJACK_PRIVATE_WRAP						Х	
CKM_SKIPJACK_RELAYX						X ³	
CKM_BATON_KEY_GEN					Х		
CKM_BATON_ECB128	X						
CKM_BATON_ECB96	X						
CKM_BATON_CBC128	X						
CKM_BATON_COUNTER	X						
CKM_BATON_SHUFFLE	X						

				Function	าร		
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_BATON_WRAP						Х	
CKM_JUNIPER_KEY_GEN					Х		
CKM_JUNIPER_ECB128	Х						
CKM_JUNIPER_CBC128	Х						
CKM_JUNIPER_COUNTER	Х						
CKM_JUNIPER_SHUFFLE	Х						
CKM_JUNIPER_WRAP						Х	
CKM_MD2				Х			
CKM_MD2_HMAC_GENERAL		Х					
CKM_MD2_HMAC		Х					
CKM_MD2_KEY_DERIVATION							Х
CKM_MD5				Х			
CKM_MD5_HMAC_GENERAL		Х					
CKM_MD5_HMAC		Х					
CKM_MD5_KEY_DERIVATION							Х
CKM_RIPEMD128				Х			
CKM_RIPEMD128_HMAC_GENERAL		Х					
CKM_RIPEMD128_HMAC		Х					
CKM_RIPEMD160				Х			
CKM_RIPEMD160_HMAC_GENERAL		Х					
CKM_RIPEMD160_HMAC		Х					
CKM_FASTHASH				Х			
CKM_PBE_MD2_DES_CBC					Х		
CKM_PBE_MD5_DES_CBC					Х		
CKM_PBE_MD5_CAST_CBC					Х		
CKM_PBE_MD5_CAST3_CBC					Х		
CKM_PBE_MD5_CAST128_CBC					Х		
(CKM_PBE_MD5_CAST5_CBC)			<u> </u>	<u> </u>			
CKM_PBE_SHA1_CAST128_CBC					Х		
(CKM_PBE_SHA1_CAST5_CBC)				1			
CKM_PBE_SHA1_RC4_128				1	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_PKCS5_PBKD2					Х	,,	
CKM_KEY_WRAP_SET_OAEP						X	
CKM_KEY_WRAP_LYNKS						Х	

¹ SR = SignRecover, VR = VerifyRecover.

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

^{206 &}lt;sup>2</sup> Single-part operations only.

^{207 &}lt;sup>3</sup> Mechanism can only be used for wrapping, not unwrapping.

- 210 In general, if a mechanism makes no mention of the ulMinKeyLen and ulMaxKeyLen fields of the
- 211 CK MECHANISM INFO structure, then those fields have no meaning for that particular mechanism.

212

213

2.1 FORTEZZA timestamp

- The FORTEZZA timestamp mechanism, denoted **CKM FORTEZZA TIMESTAMP**, is a mechanism for
- 215 single-part signatures and verification. The signatures it produces and verifies are DSA digital signatures
- over the provided hash value and the current time.
- 217 It has no parameters.
- 218 Constraints on key types and the length of data are summarized in the following table. The input and
- 219 output data may begin at the same location in memory.
- 220 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

- 221 Single-part operations only
- 222 ^{2 Data length, signature length}
- 223 For this mechanism, the *ulMinKeySlze* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- specify the supported range of DSA prime sizes, in bits.

225 **2.2 KEA**

226 **2.2.1 Definitions**

- This section defines the key type "CKK_KEA" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
- 228 attribute of key objects.
- 229 Mechanisms:
- 230 CKM KEA KEY PAIR GEN
- 231 CKM KEA KEY DERIVE

2.2.2 KEA mechanism parameters

2.2.2.1 CK KEA DERIVE PARAMS; CK KEA DERIVE PARAMS PTR

233234235

236

232

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

```
237
           typedef struct CK KEA DERIVE PARAMS {
238
          CK BBOOL isSender;
239
          CK ULONG ulRandomLen;
240
          CK BYTE PTR pRandomA;
241
          CK BYTE PTR pRandomB;
242
          CK ULONG ulPublicDataLen;
243
          CK BYTE PTR pPublicData;
244
          } CK KEA DERIVE PARAMS;
```

245 246

The fields of the structure have the following meanings:

247 248 249	isSender	Option for generating the key (called a TEK). The value is CK_TRUE if the sender (originator) generates the TEK, CK_FALSE if the recipient is regenerating the TEK
250	ulRandomLen	the size of random Ra and Rb in bytes
251	pRandomA	pointer to Ra data
252	pRandomB	pointer to Rb data
253	ulPublicDataLen	other party's KEA public key size
254	pPublicData	pointer to other party's KEA public key value

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

256 2.2.3 KEA public key objects

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

261

262

263264

282

260 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 b	
CKA_SUBPRIME ^{1,3}	Big integer	Subprime q (160 bits)
CKA_BASE ^{1,3}	Big integer	Base <i>g</i> (512 to 1024 bits, in steps of 64 bits)
CKA_VALUE ^{1,4}	Big integer	Public value y

- Refer to [PKCS #11-Base] table 15 for footnotes
- The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the "KEA domain parameters".
- The following is a sample template for creating a KEA public key object:

```
265
          CK OBJECT CLASS class = CKO PUBLIC KEY;
266
          CK KEY TYPE keyType = CKK KEA;
          CK_UTF8CHAR label[] = "A KEA public key object";
267
268
          CK_BYTE prime[] = {...};
269
          CK BYTE subprime[] = {...};
270
          CK BYTE base[] = \{...\};
          CK BYTE value[] = {...};
271
272
          CK ATTRIBUTE template[] = {
273
              {CKA CLASS, &class, sizeof(class)},
274
             {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
275
             {CKA TOKEN, &true, sizeof(true)},
276
             {CKA LABEL, label, sizeof(label)-1},
277
             {CKA PRIME, prime, sizeof(prime)},
278
             {CKA SUBPRIME, subprime, sizeof(subprime)},
279
             {CKA BASE, base, sizeof(base)},
280
             {CKA VALUE, value, sizeof(value)}
281
          };
```

2.2.4 KEA private key objects

284 KEA private key objects (object class CKO_PRIVATE_KEY, key type CKK_KEA) hold KEA private keys.

The following table defines the KEA private key object attributes, in addition to the common attributes

286 defined for this object class:

Table 4, KEA Private Key Object Attributes

Attribute	Data type	Meaning
CKA_PRIME ^{1,4,6}	Big integer	Prime <i>p</i> (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 <i>g</i> (512 to 1024 bits, in steps of 64 bits)
CKA_VALUE ^{1,4,6,7}	Big integer	Private value x

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

288 289

292

293

294

295

320

283

285

287

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

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

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

```
296
           CK OBJECT CLASS class = CKO PRIVATE KEY;
297
           CK KEY TYPE keyType = CKK KEA;
298
           CK UTF8CHAR label[] = "A KEA private key object";
299
          CK BYTE subject[] = {...};
300
          CK BYTE id[] = \{123\};
301
          CK BYTE prime[] = {...};
302
          CK BYTE subprime[] = {...};
303
           CK BYTE base[] = \{...\};
304
           CK BYTE value[] = {...];
305
           CK BBOOL true = CK TRUE;
306
           CK ATTRIBUTE template[] = {
307
             {CKA CLASS, &class, sizeof(class)},
308
             {CKA KEY TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS
309
             {CKA_TOKEN, &true, sizeof(true)},
310
             {CKA LABEL, label, sizeof(label) -1},
311
             {CKA SUBJECT, subject, sizeof(subject)},
312
             {CKA ID, id, sizeof(id)},
313
             {CKA SENSITIVE, &true, sizeof(true)},
314
             {CKA DERIVE, &true, sizeof(true)},
315
             {CKA PRIME, prime, sizeof(prime)},
316
             {CKA SUBPRIME, subprime, sizeof(subprime)},
317
             {CKA BASE, base, sizeof(base)],
318
             {CKA VALUE, value, sizeof(value)}
319
           };
```

2.2.5 KEA key pair generation

The KEA key pair generation mechanism, denoted **CKM_KEA_KEY_PAIR_GEN**, generates key pairs for the Key Exchange Algorithm, as defined by NIST's "SKIPJACK and KEA Algorithm Specification Version 2.0", 29 May 1998.

324 It does not have a parameter.

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

- 327 key. Note that this version of Cryptoki does not include a mechanism for generating these KEA domain 328 parameters.
- 329 The mechanism contributes the CKA CLASS, CKA KEY TYPE and CKA VALUE attributes to the new 330 public key and the CKA_CLASS, CKA_KEY_TYPE, CKA_PRIME, CKA_SUBPRIME, CKA_BASE, and
- 331 CKA VALUE attributes to the new private key. Other attributes supported by the KEA public and private
- key types (specifically, the flags indicating which functions the keys support) may also be specified in the 332
- 333 templates for the keys, or else are assigned default initial values.
- For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the CK MECHANISM INFO structure 334
- 335 specify the supported range of KEA prime sizes, in bits.

2.2.6 KEA key derivation

336

- 337 The KEA key derivation mechanism, denoted **CKM DEA DERIVE**, is a mechanism for key derivation
- based on KEA, the Key Exchange Algorithm, as defined by NIST's "SKIPJACK and KEA Algorithm 338
- Specification Version 2.0", 29 May 1998. 339
- 340 It has a parameter, a **CK KEA DERIVE PARAMS** structure.
- 341 This mechanism derives a secret value, and truncates the result according to the CKA KEY TYPE
- attribute of the template and, if it has one and the key type supports it, the CKA VALUE LEN attribute of 342
- the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism 343
- 344 contributes the result as the CKA VALUE attribute of the new key; other attributes required by the key
- 345 type must be specified in the template.
- As defined in the Specification, KEA can be used in two different operational modes; full mode and e-mail 346
- mode. Full mode is a two-phase key derivation sequence that requires real-time parameter exchange 347
- 348 between two parties. E-mail mode is a one-phase key derivation sequence that does not require real-
- 349 time parameter exchange. By convention, e-mail mode is designated by use of a fixed value of one (1)
- for the KEA parameter R_b (pRandomB). 350
- The operation of this mechanism depends on two of the values in the supplied 351
- 352 CK KEA DERIVE PARAMS structure, as detailed in the table below. Note that in all cases, the data
- 353 buffers pointed to by the parameter structure fields pRandomA and pRandomB must be allocated by the
- caller prior to invoking **C_DeriveKey**. Also, the values pointed to by *pRandomA* and *pRandomB* are 354
- represented as Cryptoki "Big integer" data (i.e., a sequence of bytes, most significant byte first). 355
- 356 Table 5, KEA Parameter Values and Operations

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

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

357

- key. In these cases, any object template supplied as the **C_DeriveKey** *pTemplate* argument should be ignored.
- This mechanism has the following rules about key sensitivity and extractability:
 - The CKA_SENSITIVE and CKA_EXTRACTABLE attributes in the template for the new key can both be specified to be either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some default value.
 - If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_FALSE, then the derived key will as well. If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_TRUE, then the derived has its CKA_ALWAYS_SENSITIVE attribute set to the same value as its CKA_SENSITIVE attribute.
 - Similarly, if the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_FALSE, then
 the derived key will, too. If the base key has its CKA_NEVER_EXTRACTABLE attribute set to
 CK_TRUE, then the derived key has its CKA_NEVER_EXTRACTABLE attribute set to the
 opposite value from its CKA_EXTRACTABLE attribute.
- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of KEA prime sizes, in bits.
- 375 **2.3 RC2**

362

363

364

365

366

367 368

369

370

371

372

- RC2 is a block cipher which is trademarked by RSA Security. It has a variable keysizse and an additional
- parameter, the "effective number of bits in the RC2 search space", which can take on values in the range
- 378 1-1024, inclusive. The effective number of bits in the RC2 search space is sometimes specified by an
- 379 RC2 "version number"; this "version number" is *not* the same thing as the "effective number of bits",
- 380 however. There is a canonical way to convert from one to the other.

2.3.1 Definitions

- This section defines the key type "CKK_RC2" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE as attribute of key objects.
- 384 Mechanisms:
- 385 CKM_RC2_KEY_GEN
- 386 CKM_RC2_ECB
- 387 CKM_RC2_CBC
- 388 CKM RC2 MAC
- 389 CKM_RC2_MAC_GENERAL
- 390 CKM_RC2_CBC_PAD

2.3.2 RC2 secret key objects

- 392 RC2 secret key objects (object class CKO_SECRET_KEY, key type CKK_RC2) hold RC2 keys. The
- 393 following table defines the RC2 secret key object attributes, in addition to the common attributes defined
- 394 for this object class:

391

395 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

- 396 Refer to [PKCS #11-Base] table 15 for footnotes
- 397 The following is a sample template for creating an RC2 secret key object:

```
398
          CK OBJECT CLASS class = CKO SECRET KEY;
399
           CK KEY TYPE keyType = CKK RC2;
400
          CK UTF8CHAR label[] = "An RC2 secret key object";
401
          CK BYTE value[] = {...};
402
          CK BBOOL true = CK TRUE;
403
          CK ATTRIBUTE template[] = {
404
             {CKA_CLASS, &class, sizeof(class)},
405
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
406
             {CKA TOKEN, &true, sizeof(true)},
407
             {CKA LABEL, label, sizeof(label)-1},
408
             {CKA ENCRYPT, &true, sizeof(true)},
409
             {CKA VALUE, value, sizeof(value)}
410
           };
```

411 2.3.3 RC2 mechanism parameters

- 412 2.3.3.1 CK_RC2_PARAMS; CK_RC2_PARAMS_PTR
- 413 **CK_RC2_PARAMS** provides the parameters to the **CKM_RC2_ECB** and **CMK_RC2_MAC** mechanisms.
- 414 It holds the effective number of bits in the RC2 search space. It is defined as follows:

```
typedef CK_ULONG CK_RC2_PARAMS;
```

- 416 **CK_RC2_PARAMS_PTR** is a pointer to a **CK_RC2_PARAMS**.
- 417 2.3.3.2 CK RC2 CBC PARAMS; CK RC2 CBC PARAMS PTR
- CK_RC2_CBC_PARAMS is a structure that provides the parameters to the CKM_RC2_CBC and CKM_RC2_CBC PAD mechanisms. It is defined as follows:

```
typedef struct CK_RC2_CBC_PARAMS {
    CK_ULONG uleffectiveBits;
    CK_BYTE iv[8];
} CK RC2 CBC PARAMS;
```

- The fields of the structure have the following meanings:
- 425 *ulEffectiveBits* the effective number of bits in the RC2 search space
- 426 *iv* the initialization vector (IV) for cipher block chaining mode
- 428 CK_RC2_CBC_PARAMS_PTR is a pointer to a CK_RC2_CBC_PARAMS.
- 2.3.3.3 CK_RC2_MAC_GENERAL_PARAMS;
 CK_RC2_MAC_GENERAL_PARAMS_PTR
- 431 **CK_RC2_MAC_GENERAL_PARAMS** is a structure that provides the parameters to the **CKM_RC2_MAC_GENERAL** mechanism. It is defined as follows:

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

- The fields of the structure have the following meanings:
- 438 *ulEffectiveBits* the effective number of bits in the RC2 search space
- 439 *ulMacLength* length of the MAC produced, in bytes
- 440 CK RC2 MAC GENERAL PARAMS PTR is a pointer to a CK RC2 MAC GENERAL PARAMS.

441 2.3.4 RC2 key generation

- The RC2 key generation mechanism, denoted **CKM_RC2_KEY_GEN**, is a key generation mechanism for
- 443 RSA Security's block cipher RC2.
- 444 It does not have a parameter.
- The mechanism generates RC2 keys with a particular length in bytes, as specified in the
- 446 **CKA_VALUE_LEN** attribute of the template for the key.
- The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
- key. Other attributes supported by the RC2 key type (specifically, the flags indicating which functions the
- key supports) may be specified in the template for the key, or else are assigned default initial values.
- 450 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- 451 specify the supported range of RC2 key sizes, in bits.

2.3.5 RC2-ECB

- 453 RC2-ECB, denoted CKM RC2 ECB, is a mechanism for single- and multiple-part encryption and
- decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and electronic
- 455 codebook mode as defined in FIPS PUB 81.
- It has a parameter, a **CK_RC2_PARAMS**, which indicates the effective number of bits in the RC2 search
- 457 space.

- This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to
- wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 460 CKA VALUE attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes
- so that the resulting length is a multiple of eight. The output data is the same length as the padded input
- data. It does not wrap the key type, key length, or any other information about the key; the application
- 463 must convey these separately.
- 464 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 465 **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the
- 466 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
- attribute of the new key; other attributes required by the key type must be specified in the template.
- 468 Constraints on key types and the length of data are summarized in the following table:
- 469 Table 7 RC2-ECB: Key and Data Length

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

- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of RC2 effective number of bits.
- 472 **2.3.6 RC2-CBC**
- 473 RC2_CBC, denoted **CKM_RC2_CBC**, is a mechanism for single- and multiple-part encryption and
- decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and cipher-
- block chaining mode as defined in FIPS PUB 81.
- 476 It has a parameter, a **CK_RC2_CBC_PARAMS** structure, where the first field indicates the effective
- 477 number of bits in the RC2 search space, and the next field is the initialization vector for cipher block
- 478 chaining mode.
- This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to
- wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 481 **CKA_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes
- 482 so that the resulting length is a multiple of eight. The output data is the same length as the padded input
- data. It does not wrap the key type, key length, or any other information about the key; the application
- 484 must convey these separately.
- 485 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 486 **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the
- 487 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
- attribute of the new key; other attributes required by the key type must be specified in the template.
- 489 Constraints on key types and the length of data are summarized in the following table:
- 490 Table 8, RC2-CBC: Key and Data Length

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

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

2.3.7 RC2-CBC with PKCS padding

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

- 498 It has a parameter, a **CK_RC2_CBC_PARAMS** structure, where the first field indicates the effective
- number of bits in the RC2 search space, and the next field is the initialization vector.
- The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the
- 501 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
- for the **CKA_VALUE_LEN** attribute.
- 503 In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA,
- 504 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see ***MISSING
- 505 REFERENCE*** for details). The entries in the table below for data length constraints when wrapping
- and unwrapping keys do not apply to wrapping and unwrapping private keys.

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

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

511 2.3.8 General-length RC2-MAC

- 512 General-length RC2-MAC, denoted CKM_RC2_MAC_GENERAL, is a mechanism for single-and
- 513 multiple-part signatures and verification, based on RSA Security's block cipher RC2 and data
- authorization as defined in FIPS PUB 113.
- 515 It has a parameter, a **CK_RC2_MAC_GENERAL_PARAMS** structure, which specifies the effective
- 516 number of bits in the RC2 search space and the output length desired from the mechanism.
- 517 The output bytes from this mechanism are taken from the start of the final RC2 cipher block produced in
- the MACing process.
- 519 Constraints on key types and the length of data are summarized in the following table:
- 520 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

- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of RC2 effective number of bits.
- 523 **2.3.9 RC2-MAC**

529

- RC2-MAC, denoted by **CKM_RC2_MAC**, is a special case of the general-length RC2-MA mechanism
- 525 (see Section 2.3.8). Instead of taking a CK RC2 MAC GENERAL PARAMS parameter, it takes a
- 526 **CK RC2 PARAMS** parameter, which only contains the effective number of bits in the RC2 search space.
- 527 RC2-MAC always produces and verifies 4-byte MACs.
- 528 Constraints on key types and the length of data are summarized in the following table:

530 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

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

533 **2.4 RC4**

534 **2.4.1 Definitions**

- This section defines the key type "CKK_RC4" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE attribute of key objects.
- 537 Mechanisms
- 538 CKM_RC4_KEY_GEN
- 539 CKM_RC4

540 2.4.2 RC4 secret key objects

- RC4 secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_RC4**) hold RC4 keys. The
- 542 following table defines the RC4 secret key object attributes, in addition to the common attributes defined
- for this object class:
- 544 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

- Refer to [PKCS #11-Base] table 15 for footnotes
- The following is a sample template for creating an RC4 secret key object:

```
CK OBJECT CLASS class = CKO SECRET KEY;
547
548
          CK KEY TYPE keyType = CKK RC4;
549
          CK_UTF8CHAR label[] = "An RC4 secret key object";
550
          CK BYTE value[] = \{...\};
551
          CK BBOOL true - CK TRUE;
552
          CK ATTRIBUTE template[] = {
553
             {CKA_CLASS, &class, sizeof(class)},
554
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
555
             {CKA_TOKEN, &true, sizeof(true)},
556
             {CKA LABEL, label, sizeof(label)-1},
557
             {CKA ENCRYPT, &true, sizeof(true)},
             {CKA VALUE, value, sizeof(value}
558
559
          };
```

2.4.3 RC4 key generation

- The RC4 key generation mechanism, denoted **CKM_RC4_KEY_GEN**, is a key generation mechanism for RSA Security's proprietary stream cipher RC4.
- 563 It does not have a parameter.

- The mechanism generates RC4 keys with a particular length in bytes, as specified in the
- 565 **CKA_VALUE_LEN** attribute of the template for the key.
- The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
- key. Other attributes supported by the RC4 key type (specifically, the flags indicating which functions the
- key supports) may be specified in the template for the key, or else are assigned default initial values.
- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK MECHANISM INFO** structure
- specify the supported range of RC4 key sizes, in bits.

2.4.4 RC4 mechanism

- 572 RC4, denoted CKM_RC4, is a mechanism for single- and multiple-part encryption and decryption based
- on RSA Security's proprietary stream cipher RC4.
- 574 It does not have a parameter.
- 575 Constraints on key types and the length of input and output data are summarized in the following table:
- 576 Table 13, RC4: Key and Data Length

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

- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of RC4 key sizes, in bits.
- 579 **2.5 RC5**

571

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

582 **2.5.1 Definitions**

- This section defines the key type "CKK_RC5" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE attribute of key objects.
- 585 Mechanisms:
- 586 CKM_RC5_KEY_GEN
- 587 CKM_RC5_ECB
- 588 CKM RC5 CBC
- 589 CKM_RC5_MAC
- 590 CKM_RC5_MAC_GENERAL
- 591 CMK_RC5_CBC_PAD

592 2.5.2 RC5 secret key objects

- 593 RC5 secret key objects (object class **CKO SECRET KEY**, key type **CKK RC5**) hold RC5 keys. The
- following table defines the RC5 secret key object attributes, in addition to the common attributes defined
- 595 for this object class.

598

596 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

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

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;

CK_KEY_TYPE keyType = CKK_RC5;

CK_UTF8CHAR label[] = "An RC5 secret key object";

CK_BYTE value[] = {...};

CK_BBOOL true = CK_TRUE;
```

```
CK ATTRIBUTE template[] = {
605
606
             {CKA CLASS, &class, sizeof(class)},
607
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
608
             {CKA_TOKEN, &true, sizeof(true)},
609
             {CKA LABEL, label, sizeof(label)-1},
610
             {CKA ENCRYPT, &true, sizeof(true)},
611
             {CKA VALUE, value, sizeof(value)}
612
           };
```

2.5.3 RC5 mechanism parameters 613

- 2.5.3.1 CK RC5 PARAMS; CK RC5 PARAMS PTR 614
- 615 CK_RC5_PARAMS provides the parameters to the CKM_RC5_ECB and CKM_RC5_MAC mechanisms.
- It is defined as follows: 616

```
617
          typedef struct CK RC5 PARAMS {
618
             CK ULONG ulWordsize;
619
             CK ULONG ulRounds;
620
            CK RC5 PARAMS;
```

- 621 The fields of the structure have the following meanings:
- ulWordsize wordsize of RC5 cipher in bytes 622
- ulRounds number of rounds of RC5 encipherment 623
- CK_RC5_PARAMS_PTR is a pointer to a CK_RC5_PARAMS. 624
- 2.5.3.2 CK RC5 CBC PARAMS; CK RC5 CBC PARAMS PTR 625
- 626 CK RC5 CBC PARAMS is a structure that provides the parameters to the CKM RC5 CBC and CKM RC5 CBC PAD mechanisms. It is defined as follows: 627

```
628
           typedef struct CK RC5 CBC PARAMS {
629
             CK ULONG ulWordsize;
630
             CK ULONG ulRounds;
631
             CK BYTE PTR pIv;
632
             CK ULONG ullvLen;
633
           } CK RC5 CBC PARAMS;
```

634 The fields of the structure have the following meanings:

```
ulwordSize
                                     wordsize of RC5 cipher in bytes
635
```

number of rounds of RC5 encipherment ulRounds 636

Vlα pointer to initialization vector (IV) for CBC encryption 637

ullVLen length of initialization vector (must be same as 638 639

blocksize)

- 640 CK_RC5_CBC_PARAMS_PTR is a pointer to a CK_RC5_CBC_PARAMS.
- 2.5.3.3 CK RC5 MAC GENERAL PARAMS; 641
- CK_RC5_MAC_GENERAL_PARAMS_PTR 642
- 643 CK RC5 MAC GENERAL PARAMS is a structure that provides the parameters to the CKM_RC5_MAC_GENERAL mechanism. It is defined as follows: 644

```
typedef struct CK_RC5_MAC_GENERAL_PARAMS {

CK_ULONG ulWordsize;

CK_ULONG ulRounds;

CK_ULONG ulMacLength;

CK_RC5_MAC_GENERAL_PARAMS;
```

The fields of the structure have the following meanings:

651 *ulwordSize* wordsize of RC5 cipher in bytes

652 *ulRounds* number of rounds of RC5 encipherment

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

654 CK_RC5_MAC_GENERAL_PARAMS_PTR is a pointer to a CK_RC5_MAC_GENERAL_PARAMS.

2.5.4 RC5 key generation

- The RC5 key generation mechanism, denoted **CKM_RC5_KEY_GEN**, is a key generation mechanism for
- 657 RSA Security's block cipher RC5.
- 658 It does not have a parameter.

655

- The mechanism generates RC5 keys with a particular length in bytes, as specified in the
- 660 **CKA_VALUE_LEN** attribute of the template for the key.
- The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
- key. Other attributes supported by the RC5 key type (specifically, the flags indicating which functions the
- key supports) may be specified in the template for the key, or else are assigned default initial values.
- For this mechanism, the *ulMinKeySlze* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- specify the supported range of RC5 key sizes, in bytes.

666 **2.5.5 RC5-ECB**

- RC5-ECB, denoted **CKM_RC5_ECB**, is a mechanism for single- and multiple-part encryption and
- decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and electronic
- 669 codebook mode as defined in FIPS PUB 81.
- 670 It has a parameter, CK RC5 PARAMS, which indicates the wordsize and number of rounds of
- encryption to use.
- This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to
- 673 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 674 **CKA_VALUE** attribute of the key that is wrapped, padded on the trailing end with null bytes so that the
- 675 resulting length is a multiple of the cipher blocksize (twice the wordsize). The output data is the same
- length as the padded input data. It does not wrap the key type, key length, or any other information about
- the key; the application must convey these separately.
- For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 679 CKA KEY TYPE attributes of the template and, if it has one, and the key type supports it, the
- 680 CKA VALUE LEN attribute of the template. The mechanism contributes the result as the CKA VALUE
- attribute of the new key; other attributes required by the key type must be specified in the template.
- 682 Constraints on key types and the length of data are summarized in the following table:
- 683 Table 15, RC5-ECB Key and Data Length

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

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

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

2.5.6 RC5-CBC

686

704

705

706

- RC5-CBC, denoted **CKM_RC5_CBC**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and cipher-block chaining mode as defined in FIPS PUB 81.
- It has a parameter, a **CK_RC5_CBC_PARAMS** structure, which specifies the wordsize and number of rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.
- This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the CKA_VALUE attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes so that the resulting length is a multiple of eight. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key; the application must convey these separately.
- For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the

 CKA_KEY_TYPE attribute for the template, and, if it has one, and the key type supports it, the

 CKA_VALUE_LEN attribute of the template. The mechanism contributes the result as the CKA_VALUE

 attribute of the new key; other attributes required by the key type must be specified in the template.
- 702 Constraints on key types and the length of data are summarized in the following table:
- 703 Table 16, RC5-CBC Key and Data Length

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

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

2.5.7 RC5-CBC with PKCS padding

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

- 711 It has a parameter, a CK_RC5_CBC_PARAMS structure, which specifies the wordsize and number of
- 712 rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.
- 713 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the
- 714 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
- 715 for the **CKA VALUE LEN** attribute.
- 716 In addition to being able to wrap an unwrap secret keys, this mechanism can wrap and unwrap RSA,
- 717 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section
- 718 ***MISSING REFERENCE*** for details). The entries in the table below for data length constraints when
- 719 wrapping and unwrapping keys do not apply to wrapping and unwrapping private keys.
- 720 Constraints on key types and the length of data are summarized in the following table:
- 721 Table 17, RC5-CBC with PKCS Padding; Key and Data Length

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

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

2.5.8 General-length RC5-MAC

- General-length RC5-MAC, denoted **CKM_RC5_MAC_GENERAL**, is a mechanism for single- and
- 726 multiple-part signatures and verification, based on RSA Security's block cipher RC5 and data
- authentication as defined in FIPS PUB 113.
- 728 It has a parameter, a CK_RC5_MAC_GENERAL_PARAMS structure, which specifies the wordsize and
- 729 number of rounds of encryption to use and the output length desired from the mechanism.
- The output bytes from this mechanism are taken from the start of the final RC5 cipher block produced in
- 731 the MACing process.

- 732 Constraints on key types and the length of data are summarized in the following table:
- 733 Table 18, General-length RC2-MAC: Key and Data Length

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

- For this mechanism, the *ulMinKeySize* and *ulMaxKeySlze* fields of the **CK_MECHANISM_INFO** structure specify the supported range of RC5 key sizes, in bytes.
- 736 **2.5.9 RC5-MAC**
- 737 RC5-MAC, denoted by **CKM_RC5_MAC**, is a special case of the general-length RC5-MAC mechanism.
- 738 Instead of taking a CK_RC5_MAC_GENERAL_PARAMS parameter, it takes a CK_RC5_PARAMS
- 739 parameter. RC5-MAC always produces and verifies MACs half as large as the RC5 blocksize.
- 740 Constraints on key types and the length of data are summarized in the following table:
- 741 Table 19, RC5-MAC: Key and Data Length

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

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

2.6 General block cipher

- 745 For brevity's sake, the mechanisms for the DES, CAST, CAST3, CAST128 (CAST5), IDEA and CDMF
- 746 block ciphers will be described together here. Each of these ciphers ha the following mechanisms, which
- 747 will be described in a templatized form.

2.6.1 Definitions

- 749 This section defines the key types "CKK DES", "CKK CAST", "CKK CAST3", "CKK CAST5"
- 750 (deprecated in v2.11), "CKK CAST128", "CKK IDEA" and "CKK CDMF" for type CK KEY TYPE as
- 751 used in the CKA KEY TYPE attribute of key objects.

752 Mechanisms:

744

- 753 CKM_DES_KEY_GEN
- 754 CKM DES ECB
- 755 CKM_DES_CBC
- 756 CKM_DES_MAC
- 757 CKM_DES_MAC_GENERAL
- 758 CKM_DES_CBC_PAD
- 759 CKM_CDMF_KEY_GEN
- 760 CKM_CDMF_ECB
- 761 CKM CDMF CBC
- 762 CKM_CDMF_MAC
- 763 CKM_CDMF_MAC_GENERAL
- 764 CKM_CDMF_CBC_PAD
- 765 CKM_DES_OFB64
- 766 CKM_DES_OFB8
- 767 CKM DES CFB64
- 768 CKM_DES_CFB8
- 769 CKM_CAST_KEY_GEN
- 770 CKM CAST ECB
- 771 CKM_CAST_CBC
- 772 CKM_CAST_MAC
- 773 CKM_CAST_MAC_GENERAL
- 774 CKM_CAST_CBC_PAD
- 775 CKM CAST3 KEY GEN
- 776 CKM CAST3 ECB
- 777 CKM_CAST3_CBC
- 778 CKM_CAST3_MAC
- 779 CKM CAST3 MAC GENERAL

```
780
           CKM_CAST3_CBC_PAD
781
           CKM_CAST5_KEY_GEN
782
           CKM_CAST128_KEY_GEN
           CKM_CAST5_ECB
783
784
           CKM CAST128 ECB
785
           CKM CAST5 CBC
786
           CKM_CAST128_CB C
           CKM_CAST5_MAC
787
788
           CKM CAST128 MAC
789
           CKM_CAST5_MAC_GENERAL
           CKM CAST128 MAC GENERAL
790
791
           CKM CAST5 CBC PAD
792
           CKM_CAST128_CBC_PAD
793
           CKM IDEA KEY GEN
794
           CKM IDEA ECB
795
           CKM IDEA MAC
796
           CKM_IDEA_MAC_GENERAL
797
           CKM_IDEA_CBC_PAD
```

2.6.2 DES secret key objects

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

802 Table 20, DES Secret Key Object

798

806

820

821

822

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

- 803 Refer to [PKCS #11-Base] table 15 for footnotes
- DES keys must always have their parity bits properly set as described in FIPS PUB 46-3. Attempting to create or unwrap a DES key with incorrect parity will return an error.
 - The following is a sample template for creating a DES secret key object:

```
807
          CK OBJECT CLASS class = CKO SECRET KEY;
808
          CK KEY TYPE keyType = CKK DES;
809
          CK_UTF8CHAR label[] = "A DES secret key object";
810
          CK BYTE value[8] = {...};
811
          CK BBOOL true = CK TRUE;
812
          CK ATTRIBUTE template[] = {
813
             {CKA CLASS, &class, sizeof(class)},
814
             {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
815
             {CKA_TOKEN, &true, sizeof(true)},
816
             {CKA LABEL, label, sizeof(label)-1},
817
             {CKA ENCRYPT, &true, sizeof(true)},
818
             {CKA VALUE, value, sizeof(value}
819
```

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

823 2.6.3 CAST secret key objects

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

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

829 830

844

827

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

```
831
          CK OBJECT CLASS class = CKO SECRET KEY;
832
          CK KEY TYPE keyType = CKK CAST;
833
          CK_UTF8CHAR label[] = "A CAST secret key object";
834
          CK BYTE value[] = \{...\};
835
          CK BBOOL true = CK TRUE;
836
          CK ATTRIBUTE template[] = {
837
             {CKA CLASS, &class, sizeof(class)},
838
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
839
             {CKA TOKEN, &true, sizeof(true)},
840
             {CKA LABEL, label, sizeof(label)-1},
841
             {CKA ENCRYPT, &true, sizeof(true)},
842
             {CKA VALUE, value, sizeof(value)}
843
```

2.6.4 CAST3 secret key objects

- 845 CAST3 secret key objects (object class CKO_SECRET_KEY, key type CKK_CAST3) hold CAST3 keys.
- The following table defines the CAST3 secret key object attributes, in addition to the common attributes defines for this object class:
- 848 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

- 849 Refer to [PKCS #11-Base] table 15 for footnotes
- The following is a sample template for creating a CAST3 secret key object:

```
851
          CK OBJECT CLASS class = CKO SECRET KEY;
852
          CK KEY TYPE keyType = CKK CAST3;
853
          CK UTF8CHAR label[] = "A CAST3 secret key object";
854
          CK BYTE value[] = {...};
855
          CK BBOOL true = CK_TRUE;
856
          CK ATTRIBUTE template[] = {
857
             {CKA CLASS, &class, sizeof(class)},
858
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
859
             {CKA TOKEN, &true, sizeof(true)},
860
             {CKA LABEL, label, sizeof(label)-1},
861
             {CKA ENCRYPT, &true, sizeof(true)},
862
             {CKA VALUE, value, sizeof(value)}
863
```

2.6.5 CAST128 (CAST5) secret key objects

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

Table 23, CAST128 (CAST5) Secret Key Object Attributes

868

869

870

884

885 886

887

904

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

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

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

```
871
          CK OBJECT CLASS class = CKO SECRET KEY;
872
          CK KEY TYPE keyType = CKK CAST128;
873
          CK UTF8CHAR label[] = "A CAST128 secret key object";
874
          CK BYTE value[] = {...};
          CK_BBOOL true = CK TRUE;
875
876
          CK ATTRIBUTE template[] = {
877
             {CKA CLASS, &class, sizeof(class)},
878
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
879
             {CKA TOKEN, &true, sizeof(true)},
880
             {CKA LABEL, label, sizeof(label)-1},
881
             {CKA ENCRYPT, &true, sizeof(true)},
882
             {CKA VALUE, value, sizeof(value)}
883
```

2.6.6 IDEA secret key objects

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

888 Table 24, IDEA Secret Key Object

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

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

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

```
891
          CK OBJECT CLASS class = CKO SECRET KEY;
892
          CK KEY TYPE keyType = CKK IDEA;
893
          CK UTF8CHAR label[] = "An IDEA secret key object";
894
          CK BYTE value [16] = \{...\};
895
          CK BBOOL true = CK TRUE;
896
          CK ATTRIBUTE template[] = {
897
             {CKA CLASS, &class, sizeof(class)},
898
             {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
899
             {CKA TOKEN, &true, sizeof(true)},
900
             {CKA LABEL, label, sizeof(label)-1},
901
             {CKA ENCRYPT, &true, sizeof(true)},
902
             {CKA VALUE, value, sizeof(value)}
903
```

2.6.7 CDMF secret key objects

906 IDEA secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_CDMF**) hold CDMF keys. The following 4907 table defines the CDMF secret key object attributes, in addition to the common attributes defines for this object class:

908 Table 25, CDMF Secret Key Object

905

913

927

929

930 931

932

933

934

935

936 937

938

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

- 909 Refer to [PKCS #11-Base] table 15 for footnotes
- CDMF keys must always have their parity bits properly set in exactly the same fashion described for DES keys in FIPS PUB 46-3. Attempting to create or unwrap a CDMF key with incorrect parity will return an error.
 - The following is a sample template for creating a CDMF secret key object:

```
914
          CK OBJECT CLASS class = CKO SECRET KEY;
915
          CK KEY TYPE keyType = CKK_CDMF;
916
          CK UTF8CHAR label[] = "A CDMF secret key object";
917
          CK BYTE value[8] = \{...\};
          CK BBOOL true = CK TRUE;
918
919
          CK ATTRIBUTE template[] = {
920
             {CKA CLASS, &class, sizeof(class)},
921
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
922
             {CKA TOKEN, &true, sizeof(true)},
923
             {CKA LABEL, label, sizeof(label)-1},
924
             {CKA ENCRYPT, &true, sizeof(true)},
925
             {CKA VALUE, value, sizeof(value)}
926
          };
```

2.6.8 General block cipher mechanism parameters

928 2.6.8.1 CK MAC GENERAL PARAMS; CK MAC GENERAL PARAMS PTR

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

```
typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
```

CK MAC GENERAL PARAMS PTR is a pointer to a CK MAC GENERAL PARAMS.

2.6.9 General block cipher key generation

- 939 Cipher <NAME> has a key generation mechanism, "<NAME> key generation", denoted by
- 940 **CKM_<NAME>_KEY_GEN**.
- This mechanism does not have a parameter.
- The mechanism contributes the CKA CLASS, CKA KEY TYPE, and CKA VALUE attributes to the new
- 943 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key
- supports) may be specified in the template for the key, or else are assigned default initial values.
- 945 When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB
- 946 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits
- 947 set properly.

- 948 When DES or CDMF keys are generated, it is token-dependent whether or not it is possible for "weak" or
- 949 "semi-weak" keys to be generated. Similarly, when triple-DES keys are generated, it is token-dependent
- whether or not it is possible for any of the component DES keys to be "weak" or "semi-weak" keys.
- When CAST, CAST3, or CAST128 (CAST5) keys are generated, the template for the secret key must
- 952 specify a **CKA_VALUE_LEN** attribute.
- 953 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- may or may not be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes,
- and so for the key generation mechanisms for these ciphers, the ulMinKeySize and ulMaxKeySize fields
- of the **CK_MECHANISM_INFO** structure specify the supported range of key sizes, in bytes. For the DES,
- 957 DES3 (triple-DES), IDEA and CDMF ciphers, these fields and not used.

2.6.10 General block cipher ECB

- 959 Cipher <NAME> has an electronic codebook mechanism, "<NAME>-ECB", denoted
- 960 **CKM_<NAME>_ECB**. It is a mechanism for single- and multiple-part encryption and decryption; key
- 961 wrapping; and key unwrapping with <NAME>.
- 962 It does not have a parameter.

958

- This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to
- 964 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 965 **CKA VALUE** attribute of the key that is wrapped, padded on the trailing end with null bytes so that the
- 966 resulting length is a multiple of <NAME>'s blocksize. The output data is the same length as the padded
- 967 input data. It does not wrap the key type, key length or any other information about the key; the
- 968 application must convey these separately.
- 969 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 970 CKA KEY TYPE attribute of the template and, if it has one, and the key type supports it, the
- 971 **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE**
- attribute of the new key; other attributes required by the key must be specified in the template.
- 973 Constraints on key types and the length of data are summarized in the following table:
- 974 Table 26, General Block Cipher ECB: Key and Data Length

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

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

2.6.11 General block cipher CBC

- 981 Cipher <NAME> has a cipher-block chaining mode, "<NAME>-CBC", denoted **CKM_<NAME>_CBC**. It is
- 982 a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping
- 983 with <NAME>.

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

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

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

987

988 989

990

991

992

993

994

995

996 997

998

999

1000

1001

1002

1003 1004

1005

1006

1007 1008

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

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

2.6.12 General block cipher CBC with PCKS padding

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

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

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

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

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

1009 Table 28, General Block Cipher CBC with PKCS Padding: Key and Data Length

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

blocksize length

- 1010 For this mechanism, the *ulMinKeySlze* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- may or may not be used. The CAST, CAST3 and CAST128 (CAST5) ciphers have variable key sizes,
- 1012 and so for these ciphers, the ulMinKeySize and ulMaxKeySize fields of the CK MECHANISM INFO
- 1013 structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA, and
- 1014 CDMF ciphers, these fields are not used.

1015 2.6.13 General-length general block cipher MAC

- 1016 Cipher <NAME> has a general-length MACing mode, "General-length <NAME>-MAC", denoted
- 1017 CKM <NAME> MAC GENERAL. It is a mechanism for single-and multiple-part signatures and
- 1018 verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB
- 1019 113
- 1020 It has a parameter, a CK MAC GENERAL PARAMS, which specifies the size of the output.
- The output bytes from this mechanism are taken from the start of the final cipher block produced in the
- 1022 MACing process.
- 1023 Constraints on key types and the length of input and output data are summarized in the following table:
- 1024 Table 29, General-length General Block Cipher MAC: Key and Data Length

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

- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- may or may not be used. The CAST, CAST3, and CASt128 (CAST5) ciphers have variable key sizes,
- and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO**
- 1028 structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and
- 1029 CDMF ciphers, these fields are not used.

1030 2.6.14 General block cipher MAC

- 1031 Cipher <NAME> has a MACing mechanism, "<NAME>-MAC", denoted CKM_<NAME>_MAC. This
- mechanism is a special case of the **CKM_<NAME>_MAC_GENERAL** mechanism described above. It
- always produces an output of size half as large as <NAME>'s blocksize.
- 1034 This mechanism has no parameters.
- 1035 Constraints on key types and the length of data are summarized in the following table:
- 1036 Table 30, General Block cipher MAC: Key and Data Length

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

- 1037 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
- may or may not be used. The CAST, CAST3, and CASt128 (CAST5) ciphers have variable key sizes,
- and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO**
- 1040 structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and
- 1041 CDMF ciphers, these fields are not used.

1042 **2.7 SKIPJACK**

1043 **2.7.1 Definitions**

This section defines the key type "CKK_SKIPJACK" for type CK_KEY_TYPE as used in the CKA KEY TYPE attribute of key objects.

1046 Mechanisms:

```
1047
             CKM_SKIPJACK_KEY_GEN
1048
             CKM SKIPJACK ECB64
1049
             CKM SKIPJACK CBC64
1050
             CKM SKIPJACK OFB64
1051
             CKM_SKIPJACK_CFB64
1052
             CKM SKIPJACK CFB32
1053
             CKM_SKIPJACK_CFB16
1054
             CKM_SKIPJACK_CFB8
             CKM SKIPJACK WRAP
1055
1056
             CKM_SKIPJACK_PRIVATE_WRAP
1057
             CKM SKIPJACK RELAYX
```

2.7.2 SKIPJACK secret key objects

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

1062 Table 31, SKIPJACK Secret Key Object

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

1063 Refer to [PKCS #11-Base] table 15 for footnotes

1064 1065

1066

1083

1058

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

1067 It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK key with a specified value. Nonetheless, we provide templates for doing so.

1069 The following is a sample template for creating a SKIPJACK MEK secret key object:

```
1070
           CK OBJECT CLASS class = CKO SECRET KEY;
1071
           CK KEY TYPE keyType = CKK SKIPJACK;
           CK UTF8CHAR label[] = "A SKIPJACK MEK secret key object";
1072
1073
           CK BYTE value [12] = \{...\};
1074
           CK BBOOL true = CK TRUE;
           CK ATTRIBUTE template[] = {
1075
1076
              {CKA CLASS, &class, sizeof(class)},
1077
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1078
              {CKA_TOKEN, &true, sizeof(true)},
1079
              {CKA LABEL, label, sizeof(label)-1},
1080
              {CKA ENCRYPT, &true, sizeof(true)},
1081
              {CKA VALUE, value, sizeof(value)}
1082
            };
```

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

```
1084
           CK OBJECT CLASS class = CKO SECRET KEY;
1085
           CK KEY TYPE keyType = CKK SKIPJACK;
           CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1086
1087
           CK BYTE value[12] = \{...\};
           CK BBOOL true = CK TRUE;
1088
1089
           CK ATTRIBUTE template[] = {
1090
              {CKA CLASS, &class, sizeof(class)},
1091
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1092
              {CKA TOKEN, &true, sizeof(true)},
1093
              {CKA LABEL, label, sizeof(label)-1},
1094
              {CKA ENCRYPT, &true, sizeof(true)},
1095
              {CKA WRAP, &true, sizeof(true)},
1096
              {CKA VALUE, value, sizeof(value)}
1097
```

2.7.3 SKIPJACK Mechanism parameters

1098

1099 1100

1101

1102

1116

2.7.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR

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

```
1103
           typedef struct CK SKIPJACK PRIVATE WRAP PARAMS {
1104
              CK ULONG ulPasswordLen;
1105
              CK BYTE PTR pPassword;
1106
              CK_ULONG ulPublicDataLen;
1107
              CK BYTE PTR pPublicData;
              CK_ULONG ulPandGLen;
1108
1109
              CK_ULONG ulQLen;
1110
              CK ULONG ulRandomLen;
1111
              CK BYTE PTR pRandomA;
1112
              CK BYTE PTR pPrimeP;
              CK BYTE PTR pBaseG;
1113
1114
              CK BYTE PTR pSubprimeQ;
1115
            CK SKIPJACK PRIVATE WRAP PARAMS;
```

The fields of the structure have the following meanings:

1117	ulPasswordLen	length of the password
1118 1119	pPassword	pointer to the buffer which contains the user-supplied password
1120	ulPublicDataLen	other party's key exchange public key size
1121	pPublicData	pointer to other party's key exchange public key value
1122	ulPandGLen	length of prime and base values
1123	ulQLen	length of subprime value
1124	ulRandomLen	size of random Ra, in bytes
1125	pPrimeP	pointer to Prime, p, value
1126	pBaseG	pointer to Base, b, value

```
pSubprimeQ
                                     pointer to Subprime, q, value
1127
       CK SKIPJACK PRIVATE WRAP PARAMS PTR is a pointer to a
1128
                                     CK PRIVATE WRAP PARAMS.
1129
       2.7.3.2 CK SKIPJACK RELAYX PARAMS:
1130
1131
               CK_SKIPJACK_RELAYX_PARAMS_PTR
1132
       CK SKIPJACK RELAYX PARAMS is a structure that provides the parameters to the
1133
       CKM SKIPJACK RELAYX mechanism. It is defined as follows:
1134
           typedef struct CK SKIPJACK RELAYX PARAMS {
1135
             CK ULONG ulOldWrappedXLen;
1136
             CK BYTE PTR pOldWrappedX;
1137
             CK ULONG ulOldPasswordLen;
1138
             CK BYTE PTR pOldPassword;
1139
             CK ULONG ulOldPublicDataLen;
1140
             CK BYTE PTR pOldPublicData;
1141
             CK ULONG ulOldRandomLen;
             CK BYTE PTR pOldRandomA;
1142
1143
             CK ULONG ulNewPasswordLen;
1144
             CK BYTE PTR pNewPassword;
1145
             CK ULONG ulNewPublicDataLen;
1146
             CK BYTE PTR pNewPublicData;
1147
             CK ULONG ulNewRandomLen;
1148
             CK BYTE PTR pNewRandomA;
1149
            CK SKIPJACK RELAYX PARAMS;
1150
       The fields of the structure have the following meanings:
1151
               ulOldWrappedLen
                                     length of old wrapped key in bytes
                  pOldWrappedX
                                     pointer to old wrapper key
1152
1153
               ulOldPasswordLen
                                     length of the old password
1154
                   pOldPassword
                                     pointer to the buffer which contains the old user-supplied
                                     password
1155
              ulOldPublicDataLen
                                     old key exchange public key size
1156
1157
                  pOldPublicData
                                     pointer to old key exchange public key value
                ulOldRandomLen
                                     size of old random Ra in bytes
1158
1159
                   pOldRandomA
                                     pointer to old Ra data
1160
              ulNewPasswordLen
                                     length of the new password
                  pNewPassword
                                     pointer to the buffer which contains the new user-
1161
                                     supplied password
1162
             ulNewPublicDataLen
                                     new key exchange public key size
1163
                                     pointer to new key exchange public key value
1164
                 pNewPublicData
```

1165	ulNewRandomLen	size of new random Ra in bytes
1166	pNewRandomA	pointer to new Ra data

1167 CK_SKIPJACK_RELAYX_PARAMS_PTR is a pointer to a CK_SKIPJACK_RELAYX_PARAMS.

2.7.4 SKIPJACK key generation

- 1169 The SKIPJACK key generation mechanism, denoted **CKM_SKIPJACK_KEY_GEN**, is a key generation
- 1170 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).
- 1171 It does not have a parameter.
- 1172 The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
- 1173 key.

1168

1174 **2.7.5 SKIPJACK-ECB64**

- 1175 SKIPJACK-ECB64, denoted **CKM SKIPJACK ECB64**, is a mechanism for single- and multiple-part
- 1176 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB
- 1177 185

1191

- 1178 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cant specify a particular IV when
- 1180 encrypting. It can, of course, specify a particular IV when decrypting.
- 1181 Constraints on key types and the length of data are summarized in the following table:
- 1182 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

1183 **2.7.6 SKIPJACK-CBC64**

- 1184 SKIPJACK-CBC64, denoted **CKM_SKIPJACK_CBC64**, is a mechanism for single- and multiple-part
- encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.
- 1186 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1187 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1189 Constraints on key types and the length of data are summarized in the following table:
- 1190 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

2.7.7 SKIPJACK-OFB64

- 1192 SKIPJACK-OFB64, denoted **CKM_SKIPJACK_OFB64**, is a mechanism for single- and multiple-part
- 1193 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.
- 1194 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1195 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.

- 1197 Constraints on key types and the length of data are summarized in the following table:
- 1198 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

1199 **2.7.8 SKIPJACK-CFB64**

- SKIPJACK-CFB64, denoted **CKM_SKIPJACK_CFB64**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.
- 1202 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cannot specify a particular IV when
- 1204 encrypting. It can, of course, specify a particular IV when decrypting.
- 1205 Constraints on key types and the length of data are summarized in the following table:
- 1206 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

1207 **2.7.9 SKIPJACK-CFB32**

- 1208 SKIPJACK-CFB32, denoted **CKM_SKIPJACK_CFB32**, is a mechanism for single- and multiple-part
- encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.
- 1210 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1211 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1213 Constraints on key types and the length of data are summarized in the following table:
- 1214 Table 36, SKIPJACK-CFB32: Data and Length

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

2.7.10 SKIPJACK-CFB16

1215

- 1216 SKIPJACK-CFB16, denoted CKM_SKIPJACK_CFB16, is a mechanism for single- and multiple-part
- 1217 encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.
- 1218 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1219 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1221 Constraints on key types and the length of data are summarized in the following table:
- 1222 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
----------------------------------	----------------------	---------------

1223 **2.7.11 SKIPJACK-CFB8**

- 1224 SKIPJACK-CFB8, denoted **CKM_SKIPJACK_CFB8**, is a mechanism for single- and multiple-part
- 1225 encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.
- 1226 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1227 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1229 Constraints on key types and the length of data are summarized in the following table:
- 1230 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

1231 **2.7.12 SKIPJACK-WRAP**

- 1232 The SKIPJACK-WRAP mechanism, denoted CKM_SKIPJACK_WRAP, is used to wrap and unwrap a
- 1233 secret key (MEK). It can wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.
- 1234 It does not have a parameter.

1235 2.7.13 SKIPJACK-PRIVATE-WRAP

- 1236 The SKIPJACK-PRIVATE-WRAP mechanism, denoted **CKM_SKIPJACK_PRIVATE_WRAP**, is used to
- wrap and unwrap a private key. It can wrap KEA and DSA private keys.
- 1238 It has a parameter, a CK SKIPJACK PRIVATE WRAP PARAMS structure.

1239 2.7.14 SKIPJACK-RELAYX

- 1240 The SKIPJACK-RELAYX mechanism, denoted **CKM_SKIPJACK_RELAYX**, is used with the **C_WrapKey**
- 1241 function to "change the wrapping" on a private key which was wrapped with the SKIPJACK-PRIVATE-
- 1242 WRAP mechanism (See Section 2.7.13).
- 1243 It has a parameter, a **CK_SKIPJACK_RELAYX_PARAMS** structure.
- 1244 Although the SKIPJACK-RELAYX mechanism is used with **C WrapKey**, it differs from other key-
- 1245 wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to
- 1246 **C_WrapKey**; however for the SKIPJACK_RELAYX mechanism, the [always invalid] value 0 should be
- passed as the key handle for **C_WrapKey**, and the already-wrapped key should be passed in as part of
- 1248 the **CK_SKIPJACK_RELAYX_PARAMS** structure.

1249 **2.8 BATON**

1250 **2.8.1 Definitions**

- 1251 This section defines the key type "CKK_BATON" for type CK_KEY_TYPE as used in the
- 1252 CKA KEY TYPE attribute of key objects.
- 1253 Mechanisms:
- 1254 CKM_BATON_KEY_GEN
- 1255 CKM_BATON_ECB128
- 1256 CKM BATON ECB96

```
1257 CKM_BATON_CBC128
1258 CKM_BATON_COUNTER
1259 CKM_BATON_SHUFFLE
1260 CKM_BATON_WRAP
```

1261 2.8.2 BATON secret key objects

- 1262 BATON secret key objects (object class CKO SECRET KEY, key type CKK BATON) hold single-length
- 1263 BATON keys. The following table defines the BATON secret key object attributes, in addition to the
- 1264 common attributes defined for this object class:
- 1265 Table 39, BATON Secret Key Object

Attribute Data type Meaning

CKA_VALUE^{1,4,6,7} Byte array Key value (always 40 bytes long)

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

1266 1267

1286

1301

- BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or unwrap a BATON key with incorrect checksum bits will return an error.
- 1270 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key with a specified value. Nonetheless, we provide templates for doing so.
- 1272 The following is a sample template for creating a BATON MEK secret key object:

```
1273
           CK OBJECT CLASS class = CKO SECRET KEY;
1274
           CK KEY TYPE keyType = CKK BATON;
1275
           CK UTF8CHAR label[] = "A BATON MEK secret key object";
1276
           CK BYTE value[40] = {...};
1277
           CK BBOOL true = CK TRUE;
1278
           CK ATTRIBUTE template[] = {
1279
              {CKA CLASS, &class, sizeof(class)},
1280
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1281
              {CKA TOKEN, &true, sizeof(true)},
1282
              {CKA LABEL, label, sizeof(label)-1},
1283
              {CKA_ENCRYPT, &true, sizeof(true)},
1284
              {CKA VALUE, value, sizeof(value)}
1285
           };
```

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

```
1287
           CK OBJECT CLASS class = CKO SECRET KEY;
1288
           CK KEY TYPE keyType = CKK BATON;
1289
           CK UTF8CHAR label[] = "A BATON TEK secret key object";
1290
           CK BYTE value[40] = {...};
1291
           CK BBOOL true = CK TRUE;
1292
           CK ATTRIBUTE template[] = {
1293
              {CKA CLASS, &class, sizeof(class)},
1294
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1295
              {CKA TOKEN, &true, sizeof(true)},
1296
              {CKA LABEL, label, sizeof(label)-1},
1297
              {CKA ENCRYPT, &true, sizeof(true)},
1298
              {CKA WRAP, &true, sizeof(true)},
1299
              {CKA VALUE, value, sizeof(value)}
1300
           };
```

2.8.3 BATON key generation

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

- 1304 It does not have a parameter.
- The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
- 1306 key

1307 **2.8.4 BATON-ECB128**

- 1308 BATON-ECB128, denoted CKM_BATON_ECB128, is a mechanism for single- and multiple-part
- 1309 encryption and decryption with BATON in 128-bit electronic codebook mode.
- 1310 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1311 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1313 Constraints on key types and the length of data are summarized in the following table:
- 1314 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

1315 **2.8.5 BATON-ECB96**

- 1316 BATON-ECB96, denoted **CKM_BATON_ECB96**, is a mechanism for single- and multiple-part encryption
- and decryption with BATON in 96-bit electronic codebook mode.
- 1318 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1321 Constraints on key types and the length of data are summarized in the following table:
- 1322 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

1323 **2.8.6 BATON-CBC128**

- 1324 BATON-CBC128, denoted CKM_BATON_CBC128, is a mechanism for single- and multiple-part
- encryption and decryption with BATON in 128-bit cipher-block chaining mode.
- 1326 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1329 Constraints on key types and the length of data are summarized in the following table:
- 1330 Table 42, BATON-CBC128

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

2.8.7 BATON-COUNTER

- 1332 BATON-COUNTER, denoted **CKM_BATON_COUNTER**, is a mechanism for single- and multiple-part
- 1333 encryption and decryption with BATON in counter mode.
- 1334 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1335 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1337 Constraints on key types and the length of data are summarized in the following table:
- 1338 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

1339 **2.8.8 BATON-SHUFFLE**

- 1340 BATON-SHUFFLE, denoted CKM_BATON_SHUFFLE, is a mechanism for single- and multiple-part
- 1341 encryption and decryption with BATON in shuffle mode.
- 1342 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1345 Constraints on key types and the length of data are summarized in the following table:
- 1346 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

1347 **2.8.9 BATON WRAP**

- 1348 The BATON wrap and unwrap mechanism, denoted **CKM_BATON_WRAP**, is a function used to wrap
- and unwrap a secret key (MEK). It can wrap and unwrap SKIPJACK, BATON and JUNIPER keys.
- 1350 It has no parameters.
- 1351 When used to unwrap a key, this mechanism contributes the CKA CLASS, CKA KEY TYPE, and
- 1352 **CKA VALUE** attributes to it.

1353 **2.9 JUNIPER**

1354 **2.9.1 Definitions**

- 1355 This section defines the key type "CKK_JUNIPER" for type CK_KEY_TYPE as used in the
- 1356 CKA_KEY_TYPE attribute of key objects.
- 1357 Mechanisms:
- 1358 CKM_JUNIPER_KEY_GEN
- 1359 CKM JUNIPER ECB128
- 1360 CKM_JUNIPER_CBC128
- 1361 CKM JUNIPER COUNTER
- 1362 CKM_JUNIPER_SHUFFLE

1363

1364

2.9.2 JUNIPER secret key objects

- JUNIPER secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_JUNIPER**) hold single-
- length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to
- the common attributes defined for this object class:
- 1368 Table 45, JUNIPER Secret Key Object

Attribute Data type Meaning

CKA_VALUE^{1,4,6,7} Byte array Key value (always 40 bytes long)

1369 Refer to [PKCS #11-Base] table 15 for footnotes

1370

1389

- JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or unwrap a BATON key with incorrect checksum bits will return an error.
- 1373 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key with a specified value. Nonetheless, we provide templates for doing so.
- 1375 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1376
           CK OBJECT CLASS class = CKO SECRET KEY;
1377
           CK KEY TYPE keyType = CKK JUNIPER;
1378
           CK UTF8CHAR label[] = "A JUNIPER MEK secret key object";
1379
           CK BYTE value [40] = {...};
1380
           CK BBOOL true = CK TRUE;
1381
           CK ATTRIBUTE template[] = {
1382
              {CKA CLASS, &class, sizeof(class)},
1383
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
              {CKA_TOKEN, &true, sizeof(true)},
1384
1385
              {CKA LABEL, label, sizeof(label)-1},
1386
              {CKA ENCRYPT, &true, sizeof(true)},
1387
              {CKA VALUE, value, sizeof(value)}
1388
           };
```

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

```
1390
           CK OBJECT CLASS class = CKO SECRET KEY;
1391
           CK KEY TYPE keyType = CKK JUNIPER;
1392
           CK UTF8CHAR label[] = "A JUNIPER TEK secret key object";
1393
           CK BYTE value [40] = {...};
1394
           CK BBOOL true = CK TRUE;
1395
           CK ATTRIBUTE template[] = {
1396
              {CKA CLASS, &class, sizeof(class)},
1397
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1398
              {CKA_TOKEN, &true, sizeof(true)},
              {CKA_LABEL, label, sizeof(label)-1},
1399
1400
              {CKA ENCRYPT, &true, sizeof(true)},
1401
              {CKA WRAP, &true, sizeof(true)},
1402
              {CKA VALUE, value, sizeof(value)}
1403
```

2.9.3 JUNIPER key generation

- The JUNIPER key generation mechanism, denoted **CKM_JUNIPER_KEY_GEN**, is a key generation mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).
- 1407 It does not have a parameter.
- The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
- 1409 key.

1404

2.9.4 JUNIPER-ECB128

1410

- 1411 JUNIPER-ECB128, denoted CKM_JUNIPER_ECB128, is a mechanism for single- and multiple-part
- encryption and decryption with JUNIPER in 128-bit electronic codebook mode.
- 1413 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1414 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1416 Constraints on key types and the length of data are summarized in the following table. For encryption
- 1417 and decryption, the input and output data (parts) may begin at the same location in memory.
- 1418 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

1419 **2.9.5 JUNIPER-CBC128**

- JUNIPER-CBC128, denoted CKM_JUNIPER_CBC128, is a mechanism for single- and multiple-part
- 1421 encryption and decryption with JUNIPER in 128-bit cipher block chaining mode.
- 1422 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1425 Constraints on key types and the length of data are summarized in the following table. For encryption
- and decryption, the input and output data (parts) may begin at the same location in memory.
- 1427 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

1428 **2.9.6 JUNIPER-COUNTER**

- 1429 JUNIPER-COUNTER, denoted CKM_JUNIPER_COUNTER, is a mechanism for single- and multiple-
- part encryption and decryption with JUNIPER in counter mode.
- 1431 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1434 Constraints on key types and the length of data are summarized in the following table. For encryption
- 1435 and decryption, the input and output data (parts) may begin at the same location in memory.
- 1436 Table 48, JUNIPER-COUNTER: Data and Length

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

2.9.7 JUNIPER-SHUFFLE

JUNIPER-SHUFFLE, denoted **CKM_JUNIPER_SHUFFLE**, is a mechanism for single- and multiple-part

1437

- 1440 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1441 value generated by the token in other words, the application cannot specify a particular IV when
- encrypting. It can, of course, specify a particular IV when decrypting.
- 1443 Constraints on key types and the length of data are summarized in the following table. For encryption
- 1444 and decryption, the input and output data (parts) may begin at the same location in memory.
- 1445 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

2.9.8 JUNIPER WRAP

- 1447 The JUNIPER wrap and unwrap mechanism, denoted **CKM_JUNIPER_WRAP**, is a function used to wrap
- and unwrap an MEK. It can wrap or unwrap SKIPJACK, BATON and JUNIPER keys.
- 1449 It has no parameters.
- 1450 When used to unwrap a key, this mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and
- 1451 **CKA_VALUE** attributes to it.
- 1452 **2.10 MD2**
- 1453 **2.10.1 Definitions**
- 1454 Mechanisms:
- 1455 CKM MD2
- 1456 CKM_MD2_HMAC
- 1457 CKM_MD2_HMAC_GENERAL
- 1458 CKM_MD2_KEY_DERIVATION
- 1459 2.10.2 MD2 digest
- The MD2 mechanism, denoted **CKM_MD2**, is a mechanism for message digesting, following the MD2
- message-digest algorithm defined in RFC 1319.
- 1462 It does not have a parameter.
- 1463 Constraints on the length of data are summarized in the following table:
- 1464 Table 50, MD2: Data Length

Function Data length Digest Length

C_Digest Any 16

1465 2.10.3 General-length MD2-HMAC

- 1466 The general-length MD2-HMAC mechanism, denoted CKM_MD2_HMAC_GENERAL, is a mechanism for
- signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it
- 1468 uses are generic secret keys.
- 1469 It has a parameter, a CK_MAC_GENERAL_PARAMS, which holds the length in bytes of the desired
- output. This length should be in the range 0-16 (the output size of MD2 is 16 bytes). Signatures (MACs)
- produced by this mechanism will be taken from the start of the full 16-byte HMAC output.

1472 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

1473 **2.10.4 MD2-HMAC**

- 1474 The MD2-HMAC mechanism, denoted CKM MD2 HMAC, is a special case of the general-length MD2-
- 1475 HMAC mechanism in Section 2.10.3.
- 1476 It has no parameter, and always produces an output of length 16.

2.10.5 MD2 key derivation

- 1478 MD2 key derivation, denoted CKM_MD2_KEY_DERIVATION, is a mechanism which provides the
- capability of deriving a secret key by digesting the value of another secret key with MD2.
- The value of the base key is digested once, and the result is used to make the value of the derived secret key.
- If no length or key type is provided in the template, then the key produced by this mechanism will be a generic secret key. Its length will be 16 bytes (the output size of MD2)..
- If no key type is provided in the template, but a length is, then the key produced by this mechanism will be a generic secret key of the specified length.
- If no length was provided in the template, but a key type is, then that key type must have a welldefined length. If it does, then the key produced by this mechanism will be of the type specified in the template. If it doesn't, an error will be returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by this mechanism will be of the specified type and length.
- 1491 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key will be set properly.
- 1493 If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.
- 1494 This mechanism has the following rules about key sensitivity and extractability:
- The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key can both be specified to be either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_FALSE, then the derived key will as well. If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_TRUE, then the derived key has its CKA_ALWAYS_SENSITIVE attribute set to the same value as its
 CKA SENSITIVE attribute.
- Similarly, if the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_FALSE, then the derived key will, too. If the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_TRUE, then the derived key has its CKA_NEVER_EXTRACTABLE attribute set to the opposite value from its CKA_EXTRACTABLE attribute.

1506 **2.11 MD5**

2.11.1 Definitions

1508 Mechanisms:

1509 CKM_MD5

1510 CKM_MD5_HMAC

1511	CKM_MD5_HMAC_GENERAL
1512	CKM_MD5_KEY_DERIVATION

1513 **2.11.2 MD5 Digest**

- 1514 The MD5 mechanism, denoted **CKM MD5**, is a mechanism for message digesting, following the MD5
- message-digest algorithm defined in RFC 1321.
- 1516 It does not have a parameter.
- 1517 Constraints on the length of input and output data are summarized in the following table. For single-part
- 1518 digesting, the data and the digest may begin at the same location in memory.
- 1519 Table 52, MD5: Data Length

Function Data length Digest length C Digest Any 16

1520 2.11.3 General-length MD5-HMAC

- 1521 The general-length MD5-HMAC mechanism, denoted **CKM_MD5_HMAC_GENERAL**, is a mechanism for
- 1522 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it
- uses are generic secret keys.
- 1524 It has a parameter, a CK_MAC_GENERAL_PARAMS, which holds the length in bytes of the desired
- output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)
- 1526 produced by this mechanism will be taken from the start of the full 16-byte HMAC output.
- 1527 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

1528 **2.11.4 MD5-HMAC**

- The MD5-HMAC mechanism, denoted **CKM_MD5_HMAC**, is a special case of the general-length MD5-
- 1530 HMAC mechanism in Section 2.11.3.
- 1531 It has no parameter, and always produces an output of length 16.

1532 **2.11.5 MD5 key derivation**

- 1533 MD5 key derivation denoted **CKM_MD5_KEY_DERIVATION**, is a mechanism which provides the
- capability of deriving a secret key by digesting the value of another secret key with MD5.
- The value of the base key is digested once, and the result is used to make the value of derived secret key.
- If no length or key type is provided in the template, then the key produced by this mechanism will be a generic secret key. Its length will be 16 bytes (the output size of MD5).
- If no key type is provided in the template, but a length is, then the key produced by this mechanism will be a generic secret key of the specified length.
- If no length was provided in the template, but a key type is, then that key type must have a welldefined length. If it does, then the key produced by this mechanism will be of the type specified in the template. If it doesn't, an error will be returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by this mechanism will be of the specified type and length.

- 1546 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key will be set
- 1547 properly.
- 1548 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.
- 1549 This mechanism has the following rules about key sensitivity and extractability.
- The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key can both be specified to either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_FALSE, then the derived key will as well. If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_TRUE, then the derived key has its CKA_ALWAYS_SENSITIVE attribute set to the same value as its
 CKA SENSITIVE attribute.
- Similarly, if the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_FALSE, then the derived key will, too. If the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_TRUE, then the derived key has its CKA_NEVER_EXTRACTABLE attribute set to the *opposite* value from its CKA_EXTRACTABLE attribute.

1561 **2.12 FASTHASH**

1562 **2.12.1 Definitions**

- 1563 Mechanisms:
- 1564 CKM FASTHASH

1565 2.12.2 FASTHASH digest

- 1566 The FASTHASH mechanism, denoted **CKM FASTHASH**, is a mechanism for message digesting,
- 1567 following the U.S. government's algorithm.
- 1568 It does not have a parameter.
- 1569 Constraints on the length of input and output data are summarized in the following table:
- 1570 Table 54, FASTHASH: Data Length

Function Input length Digest lengthC Digest Any 40

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

- The mechanisms in this section are for generating keys and IVs for performing password-based
- 1573 encryption. The method used to generate keys and IVs is specified in PKCS #5.

2.13.1 Definitions

1	575	Mechanisms:
	575	wechanisms.

1576	CKM_PBE_MD2_DES_CBC
1577	CKM_PBE_MD5_DES_CBC
1578	CKM_PBE_MD5_CAST_CBC
1579	CKM_PBE_MD5_CAST3_CBC
1580	CKM_PBE_MD5_CAST5_CBC
1581	CKM_PBE_MD5_CAST128_CBC
1582	CKM_PBE_SHA1_CAST5_CBC
1583	CKM_PBE_SHA1_CAST128_CBC

```
1584
            CKM_PBE_SHA1_RC4_128
1585
            CKM_PBE_SHA1_RC4_40
1586
            CKM_PBE_SHA1_RC2_128_CBC
1587
            CKM_PBE_SHA1_RC2_40_CBC
```

2.13.2 Password-based encryption/authentication mechanism parameters 1588

2.13.2.1 CK PBE PARAMS; CK PBE PARAMS PTR 1589

1590 CK PBE PARAMS is a structure which provides all of the necessary information required by the 1591 CKM PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation 1592 mechanisms) and the CKM PBA SHA1 WITH SHA1 HMAC mechanism. It is defined as follows:

```
typedef struct CK PBE PARAMS {
1593
1594
              CK BYTE PTR pInitVector;
1595
              CK UTF8CHAR PTR pPassword;
1596
              CK ULONG ulPasswordLen;
1597
              CK BYTE PTR pSalt;
1598
              CK ULONG ulSaltLen;
1599
              CK ULONG ulIteration;
             CK PBE PARAMS;
1600
```

naintar to the leastion that receives the Q buts

The fields of the structure have the following meanings: nlnit\/aatar

1602 1603	pinitVector	initialization vector (IV), if an IV is required
1604 1605	pPassword	points to the password to be used in the PBE key generation
1606	ulPasswordLen	length in bytes of the password information
1607	pSalt	points to the salt to be used in the PBE key generation
1608	ulSaltLen	length in bytes of the salt information
1609	ullteration	number of iterations required for the generation

1610 CK_PBE_PARAMS_PTR is a pointer to a CK_PBE_PARAMS.

2.13.3 MD2-PBE for DES-CBC 1611

- MD2-PBE for DES-CBC, denoted **CKM PBE MD2 DES CBC**, is a mechanism used for generating a 1612
- DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an 1613
- iteration count. This functionality is defined in PKCS #5 as PBKDF1. 1614
- It has a parameter, a CK PBE PARAMS structure. The parameter specifies the input information for the 1615
- key generation process and the location of the application-supplied buffer which will receive the 8-byte IV 1616
- 1617 generated by the mechanism.

1601

4000

1618

2.13.4 MD5-PBE for DES-CBC

- MD5-PBE for DES-CBC, denoted CKM PBE MD5 DES CBC, is a mechanism used for generating a 1619
- 1620 DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
- 1621 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

- 1622 It has a parameter, a CK_PBE_PARAMS structure. The parameter specifies the input information for the
- key generation process and the location of the application-supplied buffer which will receive the 8-byte IV
- 1624 generated by the mechanism.

1625 **2.13.5 MD5-PBE for CAST-CBC**

- MD5-PBE for CAST-CBC, denoted **CKM_PBE_MD5_CAST_CBC**, is a mechanism used for generating a
- 1627 CAST secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
- 1628 iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.
- 1629 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
- 1630 key generation process and the location of the application-supplied buffer which will receive the 8-byte IV
- 1631 generated by the mechanism
- 1632 The length of the CAST key generated by this mechanism may be specified in the supplied template; if it
- is not present in the template, it defaults to 8 bytes.

1634 **2.13.6 MD5-PBE for CAST3-CBC**

- 1635 MD5-PBE for CAST3-CBC, denoted **CKM_PBE_MD5_CAST3_CBC**, is a mechanism used for generating
- 1636 a CAST3 secret key and an IV from a password and a salt value by using the MD5 digest algorithm and
- an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.
- 1638 It has a parameter, a CK_PBE_PARAMS structure. The parameter specifies the input information for the
- key generation process and the location of the application-supplied buffer which will receive the 8-byte IV
- 1640 generated by the mechanism
- The length of the CAST3 key generated by this mechanism may be specified in the supplied template; if it
- is not present in the template, it defaults to 8 bytes.

1643 **2.13.7 MD5-PBE for CAST128-CBC (CAST5-CBC)**

- MD5-PBE for CAST128-CBC (CAST5-CBC), denoted CKM_PBE_MD5_CAST128_CBC or
- 1645 **CKM_PBE_MD5_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
- 1646 and an IV from a password and a salt value by using the MD5 digest algorithm and an iteration count.
- 1647 This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.
- 1648 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
- 1649 key generation process and the location of the application-supplied buffer which will receive the 8-byte IV
- 1650 generated by the mechanism
- The length of the CAST128 (CAST5) key generated by this mechanism may be specified in the supplied
- template; if it is not present in the template, it defaults to 8 bytes.

1653 2.13.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)

- 1654 SHA-1-PBE for CAST128-CBC (CAST5-CBC), denoted CKM_PBE_SHA1_CAST128_CBC or
- 1655 **CKM_PBE_SHA1_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
- 1656 and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This
- 1657 functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.
- 1658 It has a parameter, a **CK PBE PARAMS** structure. The parameter specifies the input information for the
- key generation process and the location of the application-supplied buffer which will receive the 8-byte IV
- 1660 generated by the mechanism
- 1661 The length of the CAST128 (CAST5) key generated by this mechanism may be specified in the supplied
- template; if it is not present in the template, it defaults to 8 bytes

2.14 PKCS #12 password-based encryption/authentication mechanisms

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

We specify here a general method for producing various types of pseudo-random bits from a password, p; a string of salt bits, s; and an iteration count, c. The "type" of pseudo-random bits to be produced is identified by an identification byte, *ID*, the meaning of which will be discussed later.

- Let H be a hash function built around a compression function $\int : \mathbf{Z}_{2}^{u} \times \mathbf{Z}_{2}^{v} \rightarrow \mathbf{Z}_{2}^{u}$ (that is, H has a chaining variable and output of length u bits, and the message input to the compression function of H is v bits). For MD2 and MD5, u=128 and v=512; for SHA-1, u=160 and v=512.
- We assume here that *u* and *v* are both multiples of 8, as are the lengths in bits of the password and salt strings and the number *n* of pseudo-random bits required. In addition, *u* and *v* are of course nonzero.
 - 1. Construct a string, *D* (the "diversifier"), by concatenating *v*/8 copies of *ID*.
 - 2. Concatenate copies of the salt together to create a string S of length $v \cdot |s/v|$ bits (the final copy of the salt may be truncated to create S). Note that if the salt is the empty string, then so is S
 - 3. Concatenate copies of the password together to create a string P of length $v \cdot |p/v|$ bits (the final copy of the password may be truncated to create P). Note that if the password is the empty string, then so is P.
 - 4. Set I=S||P| to be the concatenation of S and P.
 - Set *j*=| *n*/*u*|.

1663

1664

1676

1677

1678

1679 1680

1681 1682

1683 1684

1685

1686 1687

1688

1689

1690

1691 1692

1693

1694

1702

- 6. For i=1, 2, ..., j, do the following:
 - a. Set $A = H_c(D||I)$, the cth hash of D||I. That is, compute the hash of D||I; compute the hash of that hash; etc.; continue in this fashion until a total of c hashes have been computed, each on the result of the previous hash.
 - b. Concatenate copies of Ai to create a string B of length v bits (the final copy of Ai may be truncated to create B).
 - c. Treating *I* as a concatenation *I*0, *I*1, ..., *Ik*-1 of *v*-bit blocks, where k = s/v + p/v, modify *I* by setting $I_j = (I_j + B + 1) \mod 2v$ for each *j*. To perform this addition, treat each *v*-bit block as a binary number represented most-significant bit first
- 7. Concatenate A₁, A₂, ..., A_j together to form a pseudo-random bit string, A.
- 8. Use the first *n* bits of *A* as the output of this entire process

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

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

2.14.1 SHA-1-PBE for 128-bit RC4

- SHA-1-PBE for 128-bit RC4, denoted **CKM_PBE_SHA1_RC4_128**, is a mechanism used for generating a 128-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an iteration count. The method used to generate the key is described above.
- 1706 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the key generation process. The parameter also has a field to hold the location of an application-supplied buffer which will receive an IV; for this mechanism, the contents of this field are ignored, since RC4 does not require an IV.

- 1710 The key produced by this mechanism will typically be used for performing password-based encryption.
- 1711 2.14.2 SHA-1 PBE for 40-bit RC4
- 1712 SHA-1-PBE for 40-bit RC4, denoted **CKM_PBE_SHA1_RC4_40**, is a mechanism used for generating a
- 1713 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
- 1714 iteration count. The method used to generate the key is described above.
- 1715 It has a parameter, a CK_PBE_PARAMS structure. The parameter specifies the input information for the
- 1716 key generation process. The parameter also has a field to hold the location of an application-supplied
- buffer which will receive an IV; for this mechanism, the contents of this field are ignored, since RC4 does
- 1718 not require an IV.
- 1719 The key produced by this mechanism will typically be used for performing password-based encryption.
- 1720 2.14.3 SHA-1_PBE for 128-bit RC2-CBC
- 1721 SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_128_CBC**, is a mechanism used for
- 1722 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest
- 1723 algorithm and an iteration count. The method used to generate the key and IV is described above.
- 1724 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
- 1725 key generation process and the location of an application-supplied buffer which will receive the 8-byte IV
- 1726 generated by the mechanism.
- 1727 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
- 1728 of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object
- 1729 Identifier pbeWithSHA1And128BitRC2-CBC.
- 1730 The key and IV produced by this mechanism will typically be used for performing password-based
- 1731 encryption.
- 1732 2.14.4 SHA-1_PBE for 40-bit RC2-CBC
- 1733 SHA-1-PBE for 40-bit RC2-CBC, denoted CKM_PBE_SHA1_RC2_40_CBC, is a mechanism used for
- generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm
- 1735 and an iteration count. The method used to generate the key and IV is described above.
- 1736 It has a parameter, a CK PBE PARAMS structure. The parameter specifies the input information for the
- 1737 key generation process and the location of an application-supplied buffer which will receive the 8-byte IV
- 1738 generated by the mechanism.
- 1739 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
- of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object
- 1741 Identifier pbeWithSHA1And40BitRC2-CBC.
- 1742 The key and IV produced by this mechanism will typically be used for performing password-based
- 1743 encryption

1744 **2.15 RIPE-MD**

- 1745 **2.15.1 Definitions**
- 1746 Mechanisms:
- 1747 CKM RIPEMD128
- 1748 CKM RIPEMD128 HMAC
- 1749 CKM_RIPEMD128_HMAC_GENERAL
- 1750 CKM_RIPEMD160
- 1751 CKM_RIPEMD160_HMAC
- 1752 CKM_RIPEMD160_HMAC_GENERAL

1753 **2.15.2 RIPE-MD 128 Digest**

- 1754 The RIPE-MD 128 mechanism, denoted **CKM_RIMEMD128**, is a mechanism for message digesting,
- 1755 following the RIPE-MD 128 message-digest algorithm.
- 1756 It does not have a parameter.
- 1757 Constraints on the length of data are summarized in the following table:
- 1758 Table 55, RIPE-MD 128: Data Length

Function Data length Digest length

C_Digest Any 16

1759

1778

1760 2.15.3 General-length RIPE-MD 128-HMAC

- 1761 The general-length RIPE-MD 128-HMAC mechanism, denoted CKM_RIPEMD128_HMAC_GENERAL, is
- a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 128
- hash function. The keys it uses are generic secret keys.
- 1764 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
- output. This length should be in the range 0-16 (the output size of RIPE-MD 128 is 16 bytes). Signatures
- 1766 (MACs) produced by this mechanism will be taken from the start of the full 16-byte HMAC output.
- 1767 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

1768 **2.15.4 RIPE-MD 128-HMAC**

- 1769 The RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC**, is a special case of the
- 1770 general-length RIPE-MD 128-HMAC mechanism in Section 2.15.3.
- 1771 It has no parameter, and always produces an output of length 16.

1772 **2.15.5 RIPE-MD 160**

- 1773 The RIPE-MD 160 mechanism, denoted **CKM RIPEMD160**, is a mechanism for message digesting,
- 1774 following the RIPE-MD 160 message-digest defined in ISO-10118.
- 1775 It does not have a parameter.
- 1776 Constraints on the length of data are summarized in the following table:
- 1777 Table 57, RIPE-MD 160: Data Length

Function Data length Digest length C_Digest Any 20

2.15.6 General-length RIPE-MD 160-HMAC

- 1779 The general-length RIPE-MD 160-HMAC mechanism, denoted CKM_RIPEMD160_HMAC_GENERAL, is
- 1780 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160
- hash function. The keys it uses are generic secret keys.

- 1782 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
- 1783 output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures
- 1784 (MACs) produced by this mechanism will be taken from the start of the full 20-byte HMAC output.
- 1785 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

1786 **2.15.7 RIPE-MD 160-HMAC**

- 1787 The RIPE-MD 160-HMAC mechanism, denoted CKM_RIPEMD160_HMAC, is a special case of the
- 1788 general-length RIPE-MD 160HMAC mechanism in Section 2.15.6.
- 1789 It has no parameter, and always produces an output of length 20.
- 1790 **2.16 SET**
- 1791 **2.16.1 Definitions**
- 1792 Mechanisms:
- 1793 CKM_KEY_WRAP_SET_OAEP
- 1794 **2.16.2 SET mechanism parameters**
- 1795 2.16.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS; 1796 CK_KEY_WRAP_SET_OAEP_PARAMS_PTR
- 1797 **CK_KEY_WRAP_SET_OAEP_PARAMS** is a structure that provides the parameters to the 1798 **CKM KEY WRAP SET OAEP** mechanism. It is defined as follows:

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

1804 The fields of the structure have the following meanings:

```
bBC block contents byte

px concatenation of hash of plaintext data (if present) and extra data (if present)

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

- 1811 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR** is a pointer to a
- 1812 **CK_KEY_WRAP_SET_OAEP_PARAMS**.
- 1813 2.16.3 OAEP key wrapping for SET
- 1814 The OAEP key wrapping for SET mechanism, denoted **CKM_KEY_WRAP_SET_OAEP**, is a mechanism
- 1815 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some

- 1816 extra data may optionally be wrapped together with the DES key. This mechanism is defined in the SET
- 1817 protocol specifications.
- 1818 It takes a parameter, a **CK_KEY_WRAP_SET_OAEP_PARAMS** structure. This structure holds the
- 1819 "Block Contents" byte of the data and the concatenation of the hash of plaintext data (if present) and the
- extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by
- the *ulXLen* field having the value 0.
- When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present)
- 1823 and the extra data (if present) is returned following the convention described in Section ***MISSING
- 1824 REFERENCE*** on producing output. Note that if the inputs to **C UnwrapKey** are such that the extra
- data is not returned (e.g. the buffer supplied in the CK_KEY_WRAP_SET_OAEP_PARAMS structure is
- NULL_PTR), then the unwrapped key object will not be created, either.
- Be aware that when this mechanism is used to unwrap a key, the bBC and pX fields of the parameter
- 1828 supplied to the mechanism may be modified.
- 1829 If an application uses C_UnwrapKey with CKM_KEY_WRAP_SET_OAEP, it may be preferable for it
- simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data
- 1831 (this concatenation is never larger than 128 bytes), rather than calling **C_UnwrapKey** twice. Each call of
- 1832 C UnwrapKey with CKM KEY WRAP SET OAEP requires an RSA decryption operation to be
- performed, and this computational overhead can be avoided by this means.

1834 **2.17 LYNKS**

2.17.1 Definitions

1836 Mechanisms:

1835

1845 1846

1851

1856

1837 CKM_KEY_WRAP_LYNKS

1838 2.17.2 LYNKS key wrapping

- 1839 The LYNKS key wrapping mechanism, denoted **CKM KEY WRAP LYNKS**, is a mechanism for
- 1840 wrapping and unwrapping secret keys with DES keys. It can wrap any 8-byte secret key, and it produces
- 1841 a 10-byte wrapped key, containing a cryptographic checksum.
- 1842 It does not have a parameter.
- To wrap an 8-byte secret key K with a DES key W, this mechanism performs the following steps:
- 1844 1. Initialize two 16-bit integers, sum₁ and sum₂, to 0
 - Loop through the bytes of K from first to last.
 - 3. Set sum₁= sum₁+the key byte (treat the key byte as a number in the range 0-255).
- 1847 4. Set $sum_2 = sum_2 + sum_1$.
- 1848 5. Encrypt K with W in ECB mode, obtaining an encrypted key, E.
- 1849 6. Concatenate the last 6 bytes of *E* with sum₂, representing sum₂ most-significant bit first. The result is an 8-byte block, *T*
 - 7. Encrypt T with W in ECB mode, obtaining an encrypted checksum, C.
- 1852 8. Concatenate *E* with the last 2 bytes of *C* to obtain the wrapped key.
- When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly, an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity
- bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.

1857	3 PKCS #11 Implementation Conformance
1858 1859	An implementation is a conforming implementation if it meets the conditions specified in one or more server profiles specified in [PKCS #11-Prof] .
1860	A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.
1861 1862 1863	If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL conform to all normative statements within the clauses specified for that profile and for any subclauses to each of those clauses.
1864	

1865 Appendix A. Acknowledgments

- 1866 The following individuals have participated in the creation of this specification and are gratefully
- 1867 acknowledged:
- 1868
- 1869 **Participants**:
- 1870 Gil Abel, Athena Smartcard Solutions, Inc.
- 1871 Warren Armstrong, QuintessenceLabs
- 1872 Peter Bartok, Venafi, Inc.
- 1873 Anthony Berglas, Cryptsoft
- 1874 Kelley Burgin, National Security Agency
- 1875 Robert Burns, Thales e-Security
- 1876 Wan-Teh Chang, Google Inc.
- 1877 Hai-May Chao, Oracle
- 1878 Janice Cheng, Vormetric, Inc.
- 1879 Sangrae Cho, Electronics and Telecommunications Research Institute (ETRI)
- 1880 Doron Cohen, SafeNet, Inc.
- 1881 Fadi Cotran, Futurex
- 1882 Tony Cox, Cryptsoft
- 1883 Christopher Duane, EMC
- 1884 Chris Dunn, SafeNet, Inc.
- 1885 Valerie Fenwick, Oracle
- 1886 Terry Fletcher, SafeNet, Inc.
- 1887 Susan Gleeson, Oracle
- 1888 Sven Gossel, Charismathics
- 1889 Robert Griffin, EMC
- 1890 Paul Grojean, Individual
- 1891 Peter Gutmann, Individual
- 1892 Dennis E. Hamilton, Individual
- 1893 Thomas Hardjono, M.I.T.
- 1894 Tim Hudson, Cryptsoft
- 1895 Gershon Janssen, Individual
- 1896 Seunghun Jin, Electronics and Telecommunications Research Institute (ETRI)
- 1897 Andrey Jivsov, Symantec Corp.
- 1898 Greg Kazmierczak, Wave Systems Corp.
- 1899 Mark Knight, Thales e-Security
- 1900 Darren Krahn, Google Inc.
- 1901 Alex Krasnov, Infineon Technologies AG
- 1902 Dina Kurktchi-Nimeh, Oracle
- 1903 Mark Lambiase, SecureAuth Corporation
- 1904 Lawrence Lee, GoTrust Technology Inc.

- 1905 John Leiseboer, QuintessenceLabs
- 1906 Hal Lockhart, Oracle
- 1907 Robert Lockhart, Thales e-Security
- 1908 Dale Moberg, Axway Software
- 1909 Darren Moffat, Oracle
- 1910 Valery Osheter, SafeNet, Inc.
- 1911 Sean Parkinson, EMC
- 1912 Rob Philpott, EMC
- 1913 Mark Powers, Oracle
- 1914 Ajai Puri, SafeNet, Inc.
- 1915 Robert Relyea, Red Hat
- 1916 Saikat Saha, Oracle
- 1917 Subhash Sankuratripati, NetApp
- 1918 Johann Schoetz, Infineon Technologies AG
- 1919 Rayees Shamsuddin, Wave Systems Corp.
- 1920 Radhika Siravara, Oracle
- 1921 Brian Smith, Mozilla Corporation
- 1922 David Smith, Venafi, Inc.
- 1923 Ryan Smith, Futurex
- 1924 Jerry Smith, US Department of Defense (DoD)
- 1925 Oscar So, Oracle
- 1926 Michael Stevens, QuintessenceLabs
- 1927 Michael StJohns, Individual
- 1928 Sander Temme, Thales e-Security
- 1929 Kiran Thota, VMware, Inc.
- 1930 Walter-John Turnes, Gemini Security Solutions, Inc.
- 1931 Stef Walter, Red Hat
- 1932 Jeff Webb, Dell
- 1933 Magda Zdunkiewicz, Cryptsoft
- 1934 Chris Zimman, Bloomberg Finance L.P.

Appendix B. Manifest constants

1935

1936

1937

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

```
1938
1939
           * Copyright OASIS Open 2013. All rights reserved.
1940
           * OASIS trademark, IPR and other policies apply.
1941
           * http://www.oasis-open.org/policies-guidelines/ipr
1942
1943
1944
           #define CKK KEA 0x00000005
1945
           #define CKK RC2 0x00000011
1946
           #define CKK RC4 0x00000012
1947
           #define CKK DES 0x00000013
1948
           #define CKK CAST 0x0000016
1949
           #define CKK CAST3 0x00000017
1950
           #define CKK_CAST5 0x00000018
1951
           #define CKK_CAST128 0x00000018
1952
           #define CKK RC5 0x00000019
1953
           #define CKK IDEA 0x000001A
           #define CKK SKIPJACK 0x000001B
1954
           #define CKK BATON 0x000001C
1955
1956
           #define CKK JUNIPER 0x000001D
1957
           #define CKM MD2 RSA PKCS 0x00000004
1958
           #define CKM MD5 RSA PKCS 0x00000005
1959
           #define CKM RIPEMD128 RSA PKCS 0x00000007
1960
           #define CKM RIPEMD160 RSA PKCS 0x00000008
1961
           #define CKM RC2 KEY GEN 0x00000100
1962
           #define CKM RC2 ECB 0x00000101
1963
           #define CKM RC2 CBC 0x00000102
1964
           #define CKM RC2 MAC 0x00000103
1965
           #define CKM_RC2_MAC_GENERAL 0x00000104
1966
           #define CKM_RC2_CBC_PAD 0x00000105
1967
           #define CKM RC4 KEY GEN 0x00000110
1968
           #define CKM RC4 0x00000111
           #define CKM DES KEY GEN 0x00000120
1969
1970
           #define CKM DES ECB 0x00000121
1971
           #define CKM DES CBC 0x00000122
1972
           #define CKM DES MAC 0x00000123
1973
           #define CKM DES MAC GENERAL 0x00000124
1974
           #define CKM DES CBC PAD 0x00000125
1975
           #define CKM MD2 0x00000200
1976
           #define CKM MD2 HMAC 0x00000201
1977
           #define CKM_MD2_HMAC_GENERAL 0x00000202
1978
           #define CKM MD5 0x00000210
1979
           #define CKM MD5 HMAC 0x00000211
           #define CKM_MD5_HMAC_GENERAL 0x00000212
1980
1981
           #define CKM RIPEMD128 0x00000230
1982
           #define CKM RIPEMD128 HMAC 0x00000231
1983
           #define CKM RIPEMD128 HMAC GENERAL 0x00000232
1984
           #define CKM RIPEMD160 0x00000240
1985
           #define CKM RIPEMD160 HMAC 0x00000241
           #define CKM_RIPEMD160_HMAC_GENERAL_0x00000242
1986
1987
           #define CKM CAST KEY GEN 0x00000300
1988
           #define CKM CAST ECB 0x00000301
1989
           #define CKM CAST CBC 0x00000302
1990
           #define CKM CAST MAC 0x00000303
1991
           #define CKM CAST MAC GENERAL 0x00000304
1992
           #define CKM_CAST_CBC_PAD 0x00000305
1993
           #define CKM CAST3 KEY GEN 0x00000310
```

```
1994
            #define CKM CAST3 ECB 0x00000311
1995
            #define CKM CAST3 CBC 0x00000312
1996
            #define CKM CAST3 MAC 0x00000313
            #define CKM_CAST3_MAC_GENERAL 0x00000314
#define CKM_CAST3_CBC_PAD 0x00000315
1997
1998
            #define CKM CAST5 KEY GEN 0x00000320
1999
2000
            #define CKM CAST128 KEY GEN 0x00000320
2001
            #define CKM CAST5 ECB 0x00000321
2002
            #define CKM CAST128 ECB 0x00000321
2003
            #define CKM CAST5 CBC 0x00000322
2004
            #define CKM CAST128 CBC 0x00000322
2005
            #define CKM CAST5 MAC 0x00000323
2006
            #define CKM CAST128 MAC 0x00000323
2007
            #define CKM CAST5 MAC GENERAL 0x00000324
2008
            #define CKM CAST128 MAC GENERAL 0x00000324
2009
            #define CKM CAST5 CBC PAD 0x00000325
2010
            #define CKM CAST128 CBC PAD 0x00000325
2011
            #define CKM RC5 KEY GEN 0x00000330
2012
            #define CKM RC5 ECB 0x00000331
2013
            #define CKM RC5 CBC 0x00000332
            #define CKM RC5 MAC 0x00000333
2014
            #define CKM RC5 MAC GENERAL 0x00000334
2015
            #define CKM RC5 CBC PAD 0x00000335
2016
2017
            #define CKM IDEA KEY GEN 0x00000340
2018
            #define CKM IDEA ECB 0x00000341
2019
            #define CKM IDEA CBC 0x00000342
2020
            #define CKM IDEA MAC 0x00000343
2021
            #define CKM IDEA MAC GENERAL 0x00000344
2022
            #define CKM IDEA CBC PAD 0x00000345
            #define CKM MD5 KEY DERIVATION 0x00000390
2023
            #define CKM_MD2_KEY_DERIVATION 0x00000391
#define CKM_PBE_MD2_DES_CBC 0x000003A0
2024
2025
2026
            #define CKM PBE MD5 DES CBC 0x000003A1
2027
            #define CKM PBE MD5 CAST CBC 0x000003A2
            #define CKM PBE MD5 CAST3 CBC 0x000003A3
2028
            #define CKM PBE MD5 CAST5 CBC 0x000003A4
2029
2030
            #define CKM PBE MD5 CAST128 CBC 0x000003A4
2031
            #define CKM PBE SHA1 CAST5 CBC 0x000003A5
2032
            #define CKM PBE SHA1 CAST128 CBC 0x000003A5
2033
            #define CKM PBE SHA1 RC4 128 0x000003A6
2034
            #define CKM PBE SHA1 RC4 40 0x000003A7
2035
            #define CKM PBE SHA1 RC2 128 CBC 0x000003AA
2036
            #define CKM PBE SHA1 RC2 40 CBC 0x000003AB
2037
            #define CKM KEY WRAP LYNKS 0x00000400
2038
            #define CKM KEY WRAP SET OAEP 0x00000401
2039
            #define CKM_SKIPJACK_KEY_GEN 0x00001000
2040
            #define CKM_SKIPJACK_ECB64 0x00001001
2041
            #define CKM_SKIPJACK_CBC64 0x00001002
2042
            #define CKM SKIPJACK OFB64 0x00001003
2043
            #define CKM SKIPJACK CFB64 0x00001004
2044
            #define CKM SKIPJACK CFB32 0x00001005
2045
            #define CKM SKIPJACK CFB16 0x00001006
2046
            #define CKM SKIPJACK CFB8 0x00001007
2047
            #define CKM SKIPJACK WRAP 0x00001008
2048
            #define CKM SKIPJACK PRIVATE WRAP 0x00001009
2049
            #define CKM SKIPJACK RELAYX 0x0000100a
2050
            #define CKM KEA KEY PAIR GEN 0x00001010
            #define CKM_KEA_KEY_DERIVE 0x00001011
2051
2052
            #define CKM FORTEZZA TIMESTAMP 0x00001020
2053
            #define CKM BATON KEY GEN 0x00001030
2054
            #define CKM_BATON_ECB128 0x00001031
2055
            #define CKM_BATON_ECB96 0x00001032
2056
            #define CKM BATON CBC128 0x00001033
2057
            #define CKM BATON COUNTER 0x00001034
```

2058	#define CKM_BATON_SHUFFLE 0x00001035
2059	#define CKM BATON WRAP 0x00001036
2060	#define CKM JUNIPER KEY GEN 0x00001060
2061	#define CKM JUNIPER ECB128 0x00001061
2062	#define CKM JUNIPER CBC128 0x00001062
2063	#define CKM JUNIPER COUNTER 0x00001063
2064	#define CKM JUNIPER SHUFFLE 0x00001064
2065	#define CKM JUNIPER WRAP 0x00001065
2066	#define CKM FASTHASH 0x00001070
	_

Appendix C. Revision History

2069

2068

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

20702071