

# PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 3.0

# **OASIS Standard**

# 15 June 2020

#### This stage:

https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/os/pkcs11-hist-v3.0-os.docx (Authoritative) https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/os/pkcs11-hist-v3.0-os.html

https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/os/pkcs11-hist-v3.0-os.pdf

#### **Previous stage:**

N/A

## Latest stage:

https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/pkcs11-hist-v3.0.docx (Authoritative) https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/pkcs11-hist-v3.0.html

https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/pkcs11-hist-v3.0.pdf

#### **Technical Committee:**

OASIS PKCS 11 TC

#### **Chairs:**

Tony Cox (tony.cox@cryptsoft.com), Cryptsoft Pty Ltd Robert Relyea (rrelyea@redhat.com), Red Hat

#### **Editors:**

Chris Zimman (chris@wmpp.com), Individual

Dieter Bong (dieter.bong@utimaco.com), Utimaco IS GmbH

#### **Additional artifacts:**

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

- PKCS #11 header files:
  - https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/os/include/pkcs11-v3.0/
- ALERT: Due to a clerical error when publishing the Committee Specification, the header files listed above are outdated and may contain serious flaws. The TC is addressing this in the next round of edits. Meanwhile, users of the standard can find the correct header files at https://github.com/oasistcs/pkcs11/tree/master/working/3-00-current.

#### **Related work:**

This specification replaces or supersedes:

 PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40. Edited by Susan Gleeson, Chris Zimman, Robert Griffin, and Tim Hudson. Latest stage. http://docs.oasisopen.org/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html.

#### This specification is related to:

 PKCS #11 Cryptographic Token Interface Profiles Version 3.0. Edited by Tim Hudson. Latest stage. https://docs.oasis-open.org/pkcs11/pkcs11-profiles/v3.0/pkcs11-profiles-v3.0.html.

- PKCS #11 Cryptographic Token Interface Base Specification Version 3.0. Edited by Chris Zimman and Dieter Bong. Latest stage. https://docs.oasis-open.org/pkcs11/pkcs11-base/v3.0/pkcs11-basev3.0.html.
- PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 3.0. Edited by Chris Zimman and Dieter Bong. Latest stage. https://docs.oasis-open.org/pkcs11/pkcs11curr/v3.0/pkcs11-curr-v3.0.html.

#### **Abstract:**

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

#### Status:

This document was last revised or approved by the membership of OASIS on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc home.php?wg abbrev=pkcs11#technical.

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/pkcs11/.

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. 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 TC's web page (https://www.oasis-open.org/committees/pkcs11/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

#### **Citation format:**

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

#### [PKCS11-Historical-v3.0]

*PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 3.0.* Edited by Chris Zimman and Dieter Bong. 15 June 2020. OASIS Standard. https://docs.oasis-open.org/pkcs11-hist/v3.0/os/pkcs11-hist-v3.0-os.html. Latest stage: https://docs.oasis-open.org/pkcs11-hist/v3.0/pkcs11-hist-v3.0.html.

# **Notices**

Copyright © OASIS Open 2020. 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 <a href="https://www.oasis-open.org/policies-guidelines/trademark">https://www.oasis-open.org/policies-guidelines/trademark</a> for above quidance.

# **Table of Contents**

1	Introduction	_
	1.1 IPR Policy	8
	1.2 Description of this Document	8
	1.3 Terminology	8
	1.4 Definitions	8
	1.5 Normative References	9
	1.6 Non-Normative References	9
2	Mechanisms	12
	2.1 PKCS #11 Mechanisms	12
	2.2 FORTEZZA timestamp	14
	2.3 KEA	15
	2.3.1 Definitions	15
	2.3.2 KEA mechanism parameters	15
	2.3.2.1 CK_KEA_DERIVE_PARAMS; CK_KEA_DERIVE_PARAMS_PTR	15
	2.3.3 KEA public key objects	16
	2.3.4 KEA private key objects	16
	2.3.5 KEA key pair generation	17
	2.3.6 KEA key derivation	18
	2.4 RC2	19
	2.4.1 Definitions	19
	2.4.2 RC2 secret key objects	19
	2.4.3 RC2 mechanism parameters	
	2.4.3.1 CK_RC2_PARAMS; CK_RC2_PARAMS_PTR	
	2.4.3.2 CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR	20
	2.4.3.3 CK_RC2_MAC_GENERAL_PARAMS; CK_RC2_MAC_GENERAL_PARAMS_PTR	20
	2.4.4 RC2 key generation	21
	2.4.5 RC2-ECB	21
	2.4.6 RC2-CBC	22
	2.4.7 RC2-CBC with PKCS padding	22
	2.4.8 General-length RC2-MAC	23
	2.4.9 RC2-MAC	23
	2.5 RC4	24
	2.5.1 Definitions	24
	2.5.2 RC4 secret key objects	24
	2.5.3 RC4 key generation	24
	2.5.4 RC4 mechanism	25
	2.6 RC5	25
	2.6.1 Definitions	25
	2.6.2 RC5 secret key objects	25
	2.6.3 RC5 mechanism parameters	
	2.6.3.1 CK_RC5_PARAMS; CK_RC5_PARAMS_PTR	
	2.6.3.2 CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR	26
	2.6.3.3 CK_RC5_MAC_GENERAL_PARAMS; CK_RC5_MAC_GENERAL_PARAMS_PTR	26
	2.6.4 RC5 key generation	27

	2.6.5 RC5-ECB	. 27
	2.6.6 RC5-CBC	. 28
	2.6.7 RC5-CBC with PKCS padding	. 28
	2.6.8 General-length RC5-MAC	. 29
	2.6.9 RC5-MAC	. 29
2.	7 General block cipher	. 30
	2.7.1 Definitions	.30
	2.7.2 DES secret key objects	.31
	2.7.3 CAST secret key objects	.31
	2.7.4 CAST3 secret key objects	.32
	2.7.5 CAST128 secret key objects	.32
	2.7.6 IDEA secret key objects	. 33
	2.7.7 CDMF secret key objects	. 33
	2.7.8 General block cipher mechanism parameters	.34
	2.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR	34
	2.7.9 General block cipher key generation	. 34
	2.7.10 General block cipher ECB	. 35
	2.7.11 General block cipher CBC	. 35
	2.7.12 General block cipher CBC with PCKS padding	. 36
	2.7.13 General-length general block cipher MAC	. 36
	2.7.14 General block cipher MAC	. 37
2.	8 SKIPJACK	. 37
	2.8.1 Definitions	. 37
	2.8.2 SKIPJACK secret key objects	. 38
	2.8.3 SKIPJACK Mechanism parameters	. 39
	2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR	
	2.8.3.2 CK_SKIPJACK_RELAYX_PARAMS; CK_SKIPJACK_RELAYX_PARAMS_PTR	
	2.8.4 SKIPJACK key generation	
	2.8.5 SKIPJACK-ECB64	
	2.8.6 SKIPJACK-CBC64	
	2.8.7 SKIPJACK-OFB64	
	2.8.8 SKIPJACK-CFB64	
	2.8.9 SKIPJACK-CFB32	
	2.8.10 SKIPJACK-CFB16	
	2.8.11 SKIPJACK-CFB8	
	2.8.12 SKIPJACK-WRAP	
	2.8.13 SKIPJACK-PRIVATE-WRAP	
	2.8.14 SKIPJACK-RELAYX	
2.	9 BATON	
	2.9.1 Definitions	
	2.9.2 BATON secret key objects	
	2.9.3 BATON key generation	
	2.9.4 BATON-ECB128	
	2.9.5 BATON-ECB96	
	2.9.6 BATON-CBC128	
	2.9.7 BATON-COUNTER	.45

2.9.8 BATON-SHUFFLE	46
2.9.9 BATON WRAP	46
2.10 JUNIPER	46
2.10.1 Definitions	46
2.10.2 JUNIPER secret key objects	46
2.10.3 JUNIPER key generation	47
2.10.4 JUNIPER-ECB128	47
2.10.5 JUNIPER-CBC128	48
2.10.6 JUNIPER-COUNTER	48
2.10.7 JUNIPER-SHUFFLE	48
2.10.8 JUNIPER WRAP	49
2.11 MD2	49
2.11.1 Definitions	49
2.11.2 MD2 digest	49
2.11.3 General-length MD2-HMAC	49
2.11.4 MD2-HMAC	50
2.11.5 MD2 key derivation	50
2.12 MD5	50
2.12.1 Definitions	50
2.12.2 MD5 Digest	51
2.12.3 General-length MD5-HMAC	51
2.12.4 MD5-HMAC	51
2.12.5 MD5 key derivation	51
2.13 FASTHASH	52
2.13.1 Definitions	
2.13.2 FASTHASH digest	
2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)	
2.14.1 Definitions	
2.14.2 Password-based encryption/authentication mechanism parameters	
2.14.2.1 CK_PBE_PARAMS; CK_PBE_PARAMS_PTR	
2.14.3 MD2-PBE for DES-CBC	
2.14.4 MD5-PBE for DES-CBC	
2.14.5 MD5-PBE for CAST-CBC	
2.14.6 MD5-PBE for CAST3-CBC	
2.14.7 MD5-PBE for CAST128-CBC	
2.14.8 SHA-1-PBE for CAST128-CBC	
2.15 PKCS #12 password-based encryption/authentication mechanisms	
2.15.1 Definitions	
2.15.2 SHA-1-PBE for 128-bit RC4	
2.15.3 SHA-1_PBE for 40-bit RC4	
2.15.4 SHA-1_PBE for 128-bit RC2-CBC	
2.15.5 SHA-1_PBE for 40-bit RC2-CBC	
2.16 RIPE-MD	
2.16.1 Definitions	
2.16.2 RIPE-MD 128 Digest	
2.16.3 General-length RIPE-MD 128-HMAC	57

2.16.4 RIF	E-MD 128-HMAC	57
2.16.5 RIF	E-MD 160	57
2.16.6 Gei	neral-length RIPE-MD 160-HMAC	57
2.16.7 RIF	E-MD 160-HMAC	58
2.17 SET		58
2.17.1 Def	initions	58
2.17.2 SE	Г mechanism parameters	58
2.17.2.1	CK_KEY_WRAP_SET_OAEP_PARAMS; CK_KEY_WRAP_SET_OAEP_PARAMS_PTR	58
2.17.3 OA	EP key wrapping for SET	58
2.18 LYNKS		59
2.18.1 Def	initions	59
	NKS key wrapping	
3 PKCS #11	Implementation Conformance	60
Appendix A.	Acknowledgments	61
Appendix B.	Manifest constants	63
Appendix C.	Revision History	64

# 1 Introduction

## 2 1.1 IPR Policy

1

- 3 This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode
- 4 chosen when the Technical Committee was established. For information on whether any patents have
- 5 been disclosed that may be essential to implementing this specification, and any offers of patent licensing
- 6 terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-
- 7 open.org/committees/pkcs11/ipr.php).

# **8 1.2 Description of this Document**

- 9 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
- 11 All text is normative unless otherwise labeled.

# 12 **1.3 Terminology**

- 13 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
- 14 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described
- 15 in [RFC2119].

## 16 **1.4 Definitions**

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

18	furthor	definitions
ıΩ	IUITIEI	uemmons

19	BATON	MISSI's BATON block cipher.
20	CAST	Entrust Technologies' proprietary symmetric block cipher
21	CAST3	Entrust Technologies' proprietary symmetric block cipher
22	CAST128	Entrust Technologies' symmetric block cipher.
23 24 25	CDMF	Commercial Data Masking Facility, a block encipherment method specified by International Business Machines Corporation and based on DES.
26	CMS	Cryptographic Message Syntax (see RFC 3369)
27	DES	Data Encryption Standard, as defined in FIPS PUB 46-3
28	ECB	Electronic Codebook mode, as defined in FIPS PUB 81.
29	FASTHASH	MISSI's FASTHASH message-digesting algorithm.
30	IDEA	Ascom Systec's symmetric block cipher.
31	IV	Initialization Vector.
32	JUNIPER	MISSI's JUNIPER block cipher.
33	KEA	MISSI's Key Exchange Algorithm.
34	LYNKS	A smart card manufactured by SPYRUS.
35	MAC	Message Authentication Code
36 37	MD2	RSA Security's MD2 message-digest algorithm, as defined in RFC 6149.

38 39		MD5	RSA Security's MD5 message-digest algorithm, as defined in RFC 1321.
40		PRF	Pseudo random function.
41		RSA	The RSA public-key cryptosystem.
42		RC2	RSA Security's RC2 symmetric block cipher.
43		RC4	RSA Security's proprietary RC4 symmetric stream cipher.
44		RC5	RSA Security's RC5 symmetric block cipher.
45		SET	The Secure Electronic Transaction protocol.
46 47		SHA-1	The (revised) Secure Hash Algorithm with a 160-bit message digest, as defined in FIPS PUB 180-2.
48	S	KIPJACK	MISSI's SKIPJACK block cipher.
49			
	4 E Novembrie E	<b>D</b> = <b>f</b> = u = u =	
50	1.5 Normative F		
51 52 53	[PKCS #11-Base]	Edited by S	Cryptographic Token Interface Base Specification Version 2.40. usan Gleeson and Chris Zimman. Latest version. http://docs.oasis-ccs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html.
54 55 56 57 58	[PKCS #11-Curr]	Version 2.4	Cryptographic Token Interface Current Mechanisms Specification 0. Edited by Susan Gleeson and Chris Zimman. Latest version. pasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html.
59 60 61	[PKCS #11-Prof]	Hudson. La	Cryptographic Token Interface Profiles Version 2.40. Edited by Tim test version. http://docs.oasis-open.org/pkcs11/pkcs11-40/pkcs11-profiles-v2.40.html.
62 63 64 65	[RFC2119]		, "Key words for use in RFCs to Indicate Requirement Levels", BCP .19, March 1997. http://www.ietf.org/rfc/rfc2119.txt.
66	1.6 Non-Norma	tive Refe	rences
67	[ANSI C]	ANSI/ISO.	American National Standard for Programming Languages – C. 1990
68 69	[ANSI X9.31]		Standards Committee X9. Digital Signatures Using Reversible Public graphy for the Financial Services Industry (rDSA). 1998.
70	[ANSI X9.42]		Standards Committee X9. Public Key Cryptography for the Financial
71	-		dustry: Agreement of Symmetric Keys Using Discrete Logarithm
72 73	[ANSI X9.62]	Cryptograp Accredited	Standards Committee X9. Public Key Cryptography for the Financial
74	[/]		dustry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998
75 76 77 78	[CC/PP]	W3C. Com	Reynolds, C., H. Ohto, J. Hjelm, M. H. Butler, L. Tran, Editors, aposite Capability/Preference Profiles (CC/PP): Structure and es. 2004, URL: http://www.w3.org/TR/2004/REC-CCPP-struct-0115/
79 80	[CDPD]		Mobile Communications et al. Cellular Digital Packet Data System ons: Part 406: Airlink Security. 1993
81 82	[FIPS PUB 46-3]	NIST. FIPS	5 46-3: Data Encryption Standard (DES). October 26, 2999. URL: hist.gov/publications/fips/index.html

83	[FIPS PUB 81]	NIST. FIPS 81: DES Modes of Operation. December 1980. URL:
84	[EIDO DI ID 440]	http://csrc.nist.gov/publications/fips/index.html
85 86	[FIPS PUB 113]	NIST. FIPS 113: Computer Data Authentication. May 30, 1985. URL: http://csrc.nist.gov/publications/fips/index.html
87	[FIPS PUB 180-2]	NIST. FIPS 180-2: Secure Hash Standard. August 1, 2002. URL:
88	[FIF3 FOD 100-2]	http://csrc.nist.gov/publications/fips/index.html
89	[FORTEZZA CIPG]	NSA, Workstation Security Products. FORTEZZA Cryptologic Interface
90	<u>_</u>	Programmers Guide, Revision 1.52. November 1985
91 92	[GCS-API]	X/Open Company Ltd. Generic Cryptographic Service API (GCS-API), Base – Draft 2. February 14, 1995.
93	[ISO/IEC 7816-1]	ISO/IEC 7816-1:2011. Identification Cards – Integrated circuit cards Part 1:
94		Cards with contacts Physical Characteristics. 2011 URL:
95		http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089.
96 97	[ISO/IEC 7816-4]	ISO/IEC 7618-4:2013. Identification Cards – Integrated circuit cards – Part 4:
97 98		Organization, security and commands for interchange. 2013. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumb
99		er=54550.
100 101	[ISO/IEC 8824-1]	ISO/IEC 8824-1:2008. Abstract Syntax Notation One (ASN.1): Specification of Base Notation. 2002. URL:
102		http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=
103		54012
104	[ISO/IEC 8825-1]	ISO/IEC 8825-1:2008. Information Technology – ASN.1 Encoding Rules:
105		Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER),
106 107		and Distinguished Encoding Rules (DER). 2008. URL: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnum
108		ber=54011&ics1=35&ics2=100&ics3=60
109	[ISO/IEC 9594-1]	ISO/IEC 9594-1:2008. Information Technology – Open System Interconnection –
110		The Directory: Overview of Concepts, Models and Services. 2008. URL:
111		http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumb
112	5100 UTO 0504 03	er=53364
113 114	[ISO/IEC 9594-8]	ISO/IEC 9594-8:2008. Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks. 2008 URL:
115		http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumb
116		er=53372
117	[ISO/IEC 9796-2]	ISO/IEC 9796-2:2010. Information Technology – Security Techniques – Digital
118		Signature Scheme Giving Message Recovery – Part 2: Integer factorization
119		based mechanisms. 2010. URL:
120 121		http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumb er=54788
122	[Java MIDP]	Java Community Process. Mobile Information Device Profile for Java 2 Micro
123	[eara m.z. ]	Edition. November 2002. URL: http://jcp.org/jsr/detail/118.jsp
124	[MeT-PTD]	MeT. MeT PTD Definition – Personal Trusted Device Definition, Version 1.0.
125		February 2003. URL: http://www.mobiletransaction.org
126	[PCMCIA]	Personal Computer Memory Card International Association. PC Card Standard,
127		Release 2.1. July 1993.
128 129	[PKCS #1]	RSA Laboratories. <i>RSA Cryptography Standard, v2.1.</i> June 14, 2002 URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf
130	IDKC6 #31	RSA Laboratories. <i>Diffie-Hellman Key-Agreement Standard, v1.4.</i> November
131	[PKCS #3]	1993.
132 133	[PKCS #5]	RSA Laboratories. <i>Password-Based Encryption Standard</i> , v2.0. March 26, 1999. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf
134	[PKCS #7]	RSA Laboratories. Cryptographic Message Syntax Standard, v1.6. November
135		1997 URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v16.pdf

136 137	[PKCS #8]	RSA Laboratories. <i>Private-Key Information Syntax Standard, v1.2.</i> November 1993. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn
138 139 140	[PKCS #11-UG]	PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Edited by John Leiseboer and Robert Griffin. Latest version. http://docs.oasis-open.org/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
141 142	[PKCS #12]	RSA Laboratories. <i>Personal Information Exchange Syntax Standard, v1.0.</i> June 1999. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf
143 144 145	[RFC 1321]	R. Rivest. <i>RFC 1321: The MD5 Message-Digest Algorithm.</i> MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. URL: http://www.rfc-editor.org/rfc/rfc1321.txt
146 147	[RFC 3369]	R. Houseley. <i>RFC 3369: Cryptographic Message Syntax (CMS)</i> . August 2002. URL: http://www.rfc-editor.org/rfc/rfc3369.txt
148 149	[RFC 6149]	S. Turner and L. Chen. <i>RFC 6149: MD2 to Historic Status.</i> March, 2011. URL: http://www.rfc-editor.org/rfc/rfc6149.txt
150 151 152	[SEC-1]	Standards for Efficient Cryptography Group (SECG). Standards for Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography. Version 1.0, September 20, 2000.
153 154 155	[SEC-2]	Standards for Efficient cryptography Group (SECG). Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters. Version 1.0, September 20, 2000.
156 157	[TLS]	IETF. RFC 2246: The TLS Protocol Version 1.0. January 1999. URL: http://ieft.org/rfc/rfc2256.txt
158 159 160	[WIM]	WAP. <i>Wireless Identity Module. – WAP-260-WIP-20010712.a.</i> July 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?Doc Name=/wap/wap-260-wim-20010712-a.pdf
161 162 163 164	[WPKI]	WAP. Wireless Application Protocol: Public Key Infrastructure Definition. – WAP-217-WPKI-20010424-a. April 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?Doc Name=/wap/wap-217-wpki-20010424-a.pdf
165 166 167 168	[WTLS]	WAP. Wireless Transport Layer Security Version – WAP-261-WTLS-20010406- a. April 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?Doc Name=/wap/wap-261-wtls-20010406-a.pdf
169 170 171	[X.500]	ITU-T. Information Technology – Open Systems Interconnection –The Directory: Overview of Concepts, Models and Services. February 2001. (Identical to ISO/IEC 9594-1)
172 173 174	[X.509]	ITU-T. Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks. March 2000. (Identical to ISO/IEC 9594-8)
175 176	[X.680]	ITU-T. Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. July 2002. (Identical to ISO/IEC 8824-1)
177 178 179 180	[X.690]	ITU-T. Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). July 2002. (Identical to ISO/IEC 8825-1)

# 2 Mechanisms

## 2.1 PKCS #11 Mechanisms

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

185 186 187

188

189 190

191 192

193

181

182

183

184

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

Table 1, Mechanisms vs. Functions

Functions							
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_FORTEZZA_TIMESTAMP		X <sup>2</sup>					
CKM_KEA_KEY_PAIR_GEN					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					

	Functions							
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive	
CKM_CAST3_KEY_GEN					Х			
CKM_CAST3_ECB	X					Х		
CKM_CAST3_CBC	X					Х		
CKM_CAST3_CBC_PAD	X					Х		
CKM_CAST3_MAC_GENERAL		Х						
CKM_CAST3_MAC		Х						
CKM_CAST128_KEY_GEN					Х			
CKM_CAST128_ECB	X					Х		
CKM_CAST128_CBC	X					Х		
CKM_CAST128_CBC_PAD	X					Х		
CKM_CAST128_MAC_GENERAL		Х						
CKM_CAST128_MAC		Х						
CKM_IDEA_KEY_GEN					Х			
CKM_IDEA_ECB	Х					Х		
CKM_IDEA_CBC	Х					Х		
CKM_IDEA_CBC_PAD	X					Х		
CKM_IDEA_MAC_GENERAL		Х						
CKM_IDEA_MAC		Х						
CKM_CDMF_KEY_GEN					Х			
CKM_CDMF_ECB	X					Х		
CKM_CDMF_CBC	X					Х		
CKM_CDMF_CBC_PAD	X					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	Х							
CKM_SKIPJACK_CFB64	Х							
CKM_SKIPJACK_CFB32	X							
CKM_SKIPJACK_CFB16	Х							
CKM_SKIPJACK_CFB8	Х							
CKM_SKIPJACK_WRAP						Х		
CKM_SKIPJACK_PRIVATE_WRAP						Х		
CKM_SKIPJACK_RELAYX						X <sup>3</sup>		
CKM_BATON_KEY_GEN					Х			
CKM_BATON_ECB128	Х							
CKM_BATON_ECB96	X							
CKM_BATON_CBC128	X							
CKM_BATON_COUNTER	X							
CKM_BATON_SHUFFLE	X							
CKM_BATON_WRAP						Х	<u> </u>	
CKM_JUNIPER_KEY_GEN					Х		<u> </u>	
CKM_JUNIPER_ECB128	X			<u> </u>				
CKM_JUNIPER_CBC128	X		1	<u> </u>				

Functions							
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
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_SHA1_CAST128_CBC					Х		
CKM_PBE_SHA1_RC4_128					Х		
CKM_PBE_SHA1_RC4_40					Х		
CKM_PBE_SHA1_RC2_128_CBC					Х		
CKM_PBE_SHA1_RC2_40_CBC					Х		
CKM_PBA_SHA1_WITH_SHA1_HMAC					Х		
CKM_KEY_WRAP_SET_OAEP						Х	
CKM_KEY_WRAP_LYNKS						Х	

- 194 <sup>1</sup> SR = SignRecover, VR = VerifyRecover.
- 195 <sup>2</sup> Single-part operations only.

201

- 196 <sup>3</sup> Mechanism MUST only be used for wrapping, not unwrapping.
- The remainder of this section presents in detail the mechanisms supported by Cryptoki and the parameters which are supplied to them.
- 199 In general, if a mechanism makes no mention of the *ulMinKeyLen* and *ulMaxKeyLen* fields of the
- 200 CK\_MECHANISM\_INFO structure, then those fields have no meaning for that particular mechanism.

## 2.2 FORTEZZA timestamp

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

- 205 It has no parameters.
- 206 Constraints on key types and the length of data are summarized in the following table. The input and
- 207 output data MAY begin at the same location in memory.
- 208 Table 2, FORTEZZA Timestamp: Key and Data Length

Function	Key type	Input Length	Output Length
C_Sign <sup>1</sup>	DSA private key	20	40
C_Verify <sup>1</sup>	DSA public key	20,402	N/A

- 209 1 Single-part operations only
- 210 <sup>2 Data length, signature length</sup>
- 211 For this mechanism, the *ulMinKeySlze* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure
- specify the supported range of DSA prime sizes, in bits.
- 213 **2.3 KEA**
- 214 **2.3.1 Definitions**
- 215 This section defines the key type "CKK\_KEA" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE
- 216 attribute of key objects.
- 217 Mechanisms:

232

233

240

- 218 CKM KEA KEY PAIR GEN
- 219 CKM KEA KEY DERIVE
- 220 **2.3.2 KEA mechanism parameters**
- 221 2.3.2.1 CK KEA DERIVE PARAMS; CK KEA DERIVE PARAMS PTR
- 222 **CK\_KEA\_DERIVE\_PARAMS** is a structure that provides the parameters to the **CKM\_KEA\_DERIVE**223 mechanism. It is defined as follows:

```
224
           typedef struct CK KEA DERIVE PARAMS {
225
          CK BBOOL isSender;
226
          CK ULONG ulRandomLen;
227
          CK BYTE PTR pRandomA;
228
          CK BYTE PTR pRandomB;
229
          CK ULONG ulPublicDataLen;
230
          CK BYTE PTR pPublicData;
231
          } CK KEA DERIVE PARAMS;
```

The fields of the structure have the following meanings:

isSender Option for generating the key (called a TEK). The value 234 is CK TRUE if the sender (originator) generates the 235 TEK, CK FALSE if the recipient is regenerating the TEK 236 the size of random Ra and Rb in bytes ulRandomLen 237 pRandomA pointer to Ra data 238 pRandomB pointer to Rb data 239

ulPublicDataLen other party's KEA public key size

## 242 CK KEA DERIVE PARAMS PTR is a pointer to a CK KEA DERIVE PARAMS.

## 243 2.3.3 KEA public key objects

- 244 KEA public key objects (object class **CKO PUBLIC KEY**, key type **CKK KEA**) hold KEA public keys.
- 245 The following table defines the KEA public key object attributes, in addition to the common attributes
- 246 defined for this object class:
- 247 Table 3, KEA Public Key Object Attributes

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

- <sup>-</sup>Refer to [PKCS #11-Base] table 11 for footnotes
- The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the "KEA domain parameters".
- 251 The following is a sample template for creating a KEA public key object:

```
252
          CK OBJECT CLASS class = CKO PUBLIC KEY;
253
          CK KEY TYPE keyType = CKK KEA;
254
          CK UTF8CHAR label[] = "A KEA public key object";
255
          CK BYTE prime[] = {...};
256
          CK BYTE subprime[] = {...};
257
          CK BYTE base[] = {...};
258
          CK BYTE value[] = {...};
259
          CK ATTRIBUTE template[] = {
260
              {CKA CLASS, &class, sizeof(class)},
261
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
262
              {CKA TOKEN, &true, sizeof(true)},
263
             {CKA LABEL, label, sizeof(label)-1},
264
             {CKA PRIME, prime, sizeof(prime)},
265
             {CKA SUBPRIME, subprime, sizeof(subprime)},
266
             {CKA BASE, base, sizeof(base)},
267
             {CKA VALUE, value, sizeof(value)}
268
```

# 2.3.4 KEA private key objects

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

269

270

274 Table 4, KEA Private Key Object Attributes

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

CKA_VALUE <sup>1,4,6,7</sup> Big integ	r Private value x
--	-------------------

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

276

282

307

- 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:

```
283
          CK OBJECT CLASS class = CKO PRIVATE KEY;
284
          CK KEY TYPE keyType = CKK KEA;
285
          CK_UTF8CHAR label[] = "A KEA private key object";
286
          CK BYTE subject[] = {...};
287
          CK BYTE id[] = {123};
288
          CK BYTE prime[] = {...};
289
          CK BYTE subprime[] = {...};
290
          CK BYTE base[] = \{...\};
291
          CK BYTE value[] = {...];
292
          CK BBOOL true = CK TRUE;
293
          CK ATTRIBUTE template[] = {
294
             {CKA CLASS, &class, sizeof(class)},
295
             {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS
296
             {CKA_TOKEN, &true, sizeof(true)},
297
             {CKA LABEL, label, sizeof(label) -1},
298
             {CKA SUBJECT, subject, sizeof(subject)},
299
             {CKA ID, id, sizeof(id)},
300
             {CKA SENSITIVE, &true, sizeof(true)},
301
             {CKA DERIVE, &true, sizeof(true)},
302
             {CKA PRIME, prime, sizeof(prime)},
303
             {CKA SUBPRIME, subprime, sizeof(subprime)},
304
             {CKA BASE, base, sizeof(base)],
305
             {CKA VALUE, value, sizeof(value)}
306
```

## 2.3.5 KEA key pair generation

- The KEA key pair generation mechanism, denoted **CKM\_KEA\_KEY\_PAIR\_GEN**, generates key pairs for the Key Exchange Algorithm, as defined by NIST's "SKIPJACK and KEA Algorithm Specification Version 2.0", 29 May 1998.
- 311 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
- 314 key. Note that this version of Cryptoki does not include a mechanism for generating these KEA domain
- 315 parameters.
- The mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE and CKA\_VALUE attributes to the new
- 317 public key and the CKA\_CLASS, CKA\_KEY\_TYPE, CKA\_PRIME, CKA\_SUBPRIME, CKA\_BASE, and
- 318 **CKA\_VALUE** attributes to the new private key. Other attributes supported by the KEA public and private
- 319 key types (specifically, the flags indicating which functions the keys support) MAY also be specified in the
- templates for the keys, or else are assigned default initial values.
- 321 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure
- 322 specify the supported range of KEA prime sizes, in bits.

## 323 **2.3.6 KEA key derivation**

- The KEA key derivation mechanism, denoted **CKM\_KEA\_DERIVE**, is a mechanism for key derivation
- 325 based on KEA, the Key Exchange Algorithm, as defined by NIST's "SKIPJACK and KEA Algorithm
- 326 Specification Version 2.0", 29 May 1998.
- 327 It has a parameter, a **CK KEA DERIVE PARAMS** structure.
- 328 This mechanism derives a secret value, and truncates the result according to the CKA\_KEY\_TYPE
- 329 attribute of the template and, if it has one and the key type supports it, the CKA\_VALUE\_LEN attribute of
- the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism
- contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key
- 332 type must be specified in the template.
- 333 As defined in the Specification, KEA MAY be used in two different operational modes: full mode and e-
- mail mode. Full mode is a two-phase key derivation sequence that requires real-time parameter
- exchange between two parties. E-mail mode is a one-phase key derivation sequence that does not
- require real-time parameter exchange. By convention, e-mail mode is designated by use of a fixed value
- of one (1) for the KEA parameter  $R_b$  (*pRandomB*).
- 338 The operation of this mechanism depends on two of the values in the supplied
- 339 **CK\_KEA\_DERIVE\_PARAMS** structure, as detailed in the table below. Note that in all cases, the data
- buffers pointed to by the parameter structure fields *pRandomA* and *pRandomB* must be allocated by the
- caller prior to invoking **C\_DeriveKey**. Also, the values pointed to by *pRandomA* and *pRandomB* are
- represented as Cryptoki "Big integer" data (i.e., a sequence of bytes, most significant byte first).

343 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 <sub>a</sub> value, store it in <i>pRandomA</i> , return CKR_OK. No derived key object is created.
CK_TRUE	1	Compute KEA R <sub>a</sub> 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 <sub>a</sub> 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 <sub>b</sub> value, store it in <i>pRandomB</i> , return CKR_OK. No derived key object is created.
CK_FALSE	1	Derive key value using e-mail mode, create key object, return CKR_OK.
CK_FALSE	>1	Derive key value using full mode, create key object, return CKR_OK.

Note that the parameter value pRandomB == 0 is a flag that the KEA mechanism is being invoked to compute the party's public random value ( $R_a$  or  $R_b$ , for sender or recipient, respectively), not to derive a key. In these cases, any object template supplied as the **C\_DeriveKey** pTemplate argument should be ignored.

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

pkcs11-hist-v3.0-os Standards Track Work Product

344

345

346 347

348

<sup>\*</sup> Note that the rules regarding the CKA\_SENSITIVE, CKA\_EXTRACTABLE, CKA\_ALWAYS\_SENSITIVE, and CKA\_NEVER\_EXTRACTABLE attributes have changed in version

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
  - If the base key has its CKA\_ALWAYS\_SENSITIVE attribute set to CK\_FALSE, then the derived key MUST as well. If the base key has its CKA\_ALWAYS\_SENSITIVE attribute set to CK\_TRUE, then the derived has its CKA\_ALWAYS\_SENSITIVE attribute set to the same value as its CKA\_SENSITIVE attribute.
  - Similarly, if the base key has its CKA\_NEVER\_EXTRACTABLE attribute set to CK\_FALSE, then
    the derived key MUST, too. If the base key has its CKA\_NEVER\_EXTRACTABLE attribute set
    to CK\_TRUE, then the derived key has its CKA\_NEVER\_EXTRACTABLE attribute set to the
    opposite value from its CKA\_EXTRACTABLE attribute.
- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of KEA prime sizes, in bits.

#### 362 **2.4 RC2**

352

353

354

355

356 357

358

359

#### 363 **2.4.1 Definitions**

- 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 MAY take on values in the
- range 1-1024, inclusive. The effective number of bits in the RC2 search space is sometimes specified by
- an RC2 "version number"; this "version number" is *not* the same thing as the "effective number of bits",
- however. There is a canonical way to convert from one to the other.
- This section defines the key type "CKK\_RC2" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.
- 371 Mechanisms:
- 372 CKM RC2 KEY GEN
- 373 CKM RC2 ECB
- 374 CKM RC2 CBC
- 375 CKM RC2 MAC
- 376 CKM RC2 MAC GENERAL
- 377 CKM\_RC2\_CBC\_PAD

# 378 2.4.2 RC2 secret key objects

- 379 RC2 secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_RC2) hold RC2 keys. The
- 380 following table defines the RC2 secret key object attributes, in addition to the common attributes defined
- 381 for this object class:
- 382 Table 6, RC2 Secret Key Object Attributes

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

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

2.11 to match the policy used by other key derivation mechanisms such as **CKM SSL3 MASTER KEY DERIVE**.

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

```
385
          CK OBJECT CLASS class = CKO SECRET KEY;
386
          CK KEY TYPE keyType = CKK RC2;
387
          CK UTF8CHAR label[] = "An RC2 secret key object";
388
           CK BYTE value[] = {...};
389
          CK BBOOL true = CK TRUE;
390
          CK ATTRIBUTE template[] = {
             {CKA CLASS, &class, sizeof(class)},
391
392
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
393
             {CKA TOKEN, &true, sizeof(true)},
394
             {CKA LABEL, label, sizeof(label)-1},
395
             {CKA ENCRYPT, &true, sizeof(true)},
396
             {CKA VALUE, value, sizeof(value)}
397
```

2.4.3 RC2 mechanism parameters

398

- 399 2.4.3.1 CK\_RC2\_PARAMS; CK\_RC2\_PARAMS\_PTR
- 400 CK\_RC2\_PARAMS provides the parameters to the CKM\_RC2\_ECB and CMK\_RC2\_MAC mechanisms.
- 401 It holds the effective number of bits in the RC2 search space. It is defined as follows:

```
typedef CK_ULONG CK_RC2_PARAMS;
```

- 403 **CK\_RC2\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_PARAMS**.
- 404 2.4.3.2 CK RC2 CBC PARAMS; CK RC2 CBC PARAMS PTR
- 405 **CK\_RC2\_CBC\_PARAMS** is a structure that provides the parameters to the **CKM\_RC2\_CBC** and 406 **CKM\_RC2\_CBC PAD** mechanisms. It is defined as follows:

```
typedef struct CK_RC2_CBC_PARAMS {
    CK_ULONG uleffectiveBits;
    CK_BYTE iv[8];

410
} CK_RC2_CBC_PARAMS;
```

- The fields of the structure have the following meanings:
- 412 *ulEffectiveBits* the effective number of bits in the RC2 search space
- 413 *iv* the initialization vector (IV) for cipher block chaining mode
- 415 CK\_RC2\_CBC\_PARAMS\_PTR is a pointer to a CK\_RC2\_CBC\_PARAMS.
- 416 2.4.3.3 CK\_RC2\_MAC\_GENERAL\_PARAMS; 417 CK RC2 MAC GENERAL PARAMS PTR
- 418 **CK\_RC2\_MAC\_GENERAL\_PARAMS** is a structure that provides the parameters to the
- 419 **CKM\_RC2\_MAC\_GENERAL** mechanism. It is defined as follows:

```
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:
- 425 *ulEffectiveBits* the effective number of bits in the RC2 search space
- 426 *ulMacLength* length of the MAC produced, in bytes

- 427 CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR is a pointer to a CK\_RC2\_MAC\_GENERAL\_PARAMS.
- 428 2.4.4 RC2 key generation
- The RC2 key generation mechanism, denoted CKM\_RC2\_KEY\_GEN, is a key generation mechanism for
- 430 RSA Security's block cipher RC2.
- 431 It does not have a parameter.
- The mechanism generates RC2 keys with a particular length in bytes, as specified in the
- 433 **CKA\_VALUE\_LEN** attribute of the template for the key.
- 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.
- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure
- specify the supported range of RC2 key sizes, in bits.
- 439 **2.4.5 RC2-ECB**
- 440 RC2-ECB, denoted CKM\_RC2\_ECB, is a mechanism for single- and multiple-part encryption and
- decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and electronic
- 442 codebook mode as defined in FIPS PUB 81.
- It has a parameter, a **CK\_RC2\_PARAMS**, which indicates the effective number of bits in the RC2 search
- 444 space.
- This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to
- wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 447 **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
- 450 must convey these separately.
- 451 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 452 **CKA KEY TYPE** attribute of the template and, if it has one, and the key type supports it, the
- 453 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**
- attribute of the new key; other attributes required by the key type must be specified in the template.
- 455 Constraints on key types and the length of data are summarized in the following table:
- 456 Table 7 RC2-ECB: Key and Data Length

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

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

#### 2.4.6 RC2-CBC

- 460 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-
- 462 block chaining mode as defined in FIPS PUB 81.
- 463 It has a parameter, a CK RC2 CBC PARAMS structure, where the first field indicates the effective
- number of bits in the RC2 search space, and the next field is the initialization vector for cipher block
- 465 chaining mode.

459

- 466 This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to
- wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 468 **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
- 471 must convey these separately.
- 472 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 473 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the
- 474 CKA VALUE LEN attribute of the template. The mechanism contributes the result as the CKA VALUE
- attribute of the new key; other attributes required by the key type must be specified in the template.
- 476 Constraints on key types and the length of data are summarized in the following table:

#### 477 Table 8, RC2-CBC: Key and Data Length

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

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

## 2.4.7 RC2-CBC with PKCS padding

- 481 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
- 483 RC2; cipher-block chaining mode as defined in FIPS PUB 81; and the block cipher padding method
- detailed in PKCS #7.

480

- 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
- 488 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
- for the **CKA\_VALUE\_LEN** attribute.
- 490 In addition to being able to wrap and unwrap secret keys, this mechanism MAY wrap and unwrap RSA,
- 491 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see [PKCS #11-
- 492 Curr], Miscellaneous simple key derivation mechanisms for details). The entries in the table below
- 493 for data length constraints when wrapping and unwrapping keys do not apply to wrapping and
- 494 unwrapping private keys.
- Constraints on key types and the length of data are summarized in the following table:

#### 496 Table 9, RC2-CBC with PKCS Padding: Key and Data Length

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

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

## 2.4.8 General-length RC2-MAC

- General-length RC2-MAC, denoted **CKM\_RC2\_MAC\_GENERAL**, is a mechanism for single-and multiple-part signatures and verification, based on RSA Security's block cipher RC2 and data
- authorization as defined in FIPS PUB 113.
- It has a parameter, a **CK\_RC2\_MAC\_GENERAL\_PARAMS** structure, which specifies the effective number of bits in the RC2 search space and the output length desired from the mechanism.
- The output bytes from this mechanism are taken from the start of the final RC2 cipher block produced in the MACing process.
- 507 Constraints on key types and the length of data are summarized in the following table:
- 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.
- 511 **2.4.9 RC2-MAC**

499

- 512 RC2-MAC, denoted by **CKM RC2 MAC**, is a special case of the general-length RC2-MA mechanism
- 513 (see Section 2.4.8). Instead of taking a CK\_RC2\_MAC\_GENERAL\_PARAMS parameter, it takes a
- 514 **CK RC2 PARAMS** parameter, which only contains the effective number of bits in the RC2 search space.
- 515 RC2-MAC produces and verifies 4-byte MACs.
- 516 Constraints on key types and the length of data are summarized in the following table:
- 517 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.

## 520 **2.5 RC4**

#### 521 **2.5.1 Definitions**

- This section defines the key type "CKK\_RC4" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE
- 523 attribute of key objects.
- 524 Mechanisms
- 525 CKM\_RC4\_KEY\_GEN
- 526 CKM\_RC4

## 527 2.5.2 RC4 secret key objects

- RC4 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC4**) hold RC4 keys. The
- 529 following table defines the RC4 secret key object attributes, in addition to the common attributes defined
- 530 for this object class:
- 531 Table 12, RC4 Secret Key Object

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

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

```
534
          CK OBJECT CLASS class = CKO SECRET KEY;
535
          CK KEY TYPE keyType = CKK RC4;
536
          CK UTF8CHAR label[] = "An RC4 secret key object";
537
          CK BYTE value[] = {...};
538
          CK BBOOL true - CK TRUE;
539
          CK ATTRIBUTE template[] = {
540
             {CKA CLASS, &class, sizeof(class)},
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
541
542
             {CKA_TOKEN, &true, sizeof(true)},
543
             {CKA LABEL, label, sizeof(label)-1},
544
             {CKA ENCRYPT, &true, sizeof(true)},
545
             {CKA VALUE, value, sizeof(value}
546
          };
```

## 2.5.3 RC4 key generation

- The RC4 key generation mechanism, denoted **CKM\_RC4\_KEY\_GEN**, is a key generation mechanism for
- RSA Security's proprietary stream cipher RC4.
- 550 It does not have a parameter.

547

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

#### 2.5.4 RC4 mechanism

- 859 RC4, denoted **CKM\_RC4**, is a mechanism for single- and multiple-part encryption and decryption based
- on RSA Security's proprietary stream cipher RC4.
- It does not have a parameter.
- 562 Constraints on key types and the length of input and output data are summarized in the following table:
- 563 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.
- 566 **2.6 RC5**

558

## 567 **2.6.1 Definitions**

- RC5 is a parameterizable block cipher patented by RSA Security. It has a variable wordsize, a variable
- keysize, and a variable number of rounds. The blocksize of RC5 is equal to twice its wordsize.
- 570 This section defines the key type "CKK RC5" for type CK KEY TYPE as used in the CKA KEY TYPE
- attribute of key objects.
- 572 Mechanisms:
- 573 CKM\_RC5\_KEY\_GEN
- 574 CKM RC5 ECB
- 575 CKM RC5 CBC
- 576 CKM\_RC5\_MAC
- 577 CKM RC5 MAC GENERAL
- 578 CMK\_RC5\_CBC\_PAD

## 579 2.6.2 RC5 secret key objects

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

585

583 Table 14, RC5 Secret Key Object

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

- Refer to [PKCS #11-Base] table 11 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;
```

## 2.6.3 RC5 mechanism parameters

- 601 2.6.3.1 CK\_RC5\_PARAMS; CK\_RC5\_PARAMS\_PTR
- 602 **CK\_RC5\_PARAMS** provides the parameters to the **CKM\_RC5\_ECB** and **CKM\_RC5\_MAC** mechanisms.
- 603 It is defined as follows:

600

609

626

```
typedef struct CK_RC5_PARAMS {
    CK_ULONG ulWordsize;
    CK_ULONG ulRounds;
} CK_RC5_PARAMS;
```

The fields of the structure have the following meanings:

ulWordsize wordsize of RC5 cipher in bytes

610 *ulRounds* number of rounds of RC5 encipherment

- 611 **CK\_RC5\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_PARAMS**.
- 612 2.6.3.2 CK\_RC5\_CBC\_PARAMS; CK\_RC5\_CBC\_PARAMS\_PTR
- 613 **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:

```
typedef struct CK_RC5_CBC_PARAMS {

CK_ULONG ulWordsize;

CK_ULONG ulRounds;

CK_BYTE_PTR pIv;

CK_ULONG ulIvLen;

CK_RC5_CBC_PARAMS;
```

The fields of the structure have the following meanings:

622 *ulwordSize* wordsize of RC5 cipher in bytes

623 *ulRounds* number of rounds of RC5 encipherment

624 plV pointer to initialization vector (IV) for CBC encryption

625 *ullVLen* length of initialization vector (must be same as

blocksize)

- 627 **CK\_RC5\_CBC\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_CBC\_PARAMS**.
- 2.6.3.3 CK\_RC5\_MAC\_GENERAL\_PARAMS; 629 CK RC5 MAC GENERAL PARAMS PTR
- 630 **CK\_RC5\_MAC\_GENERAL\_PARAMS** is a structure that provides the parameters to the CKM\_RC5\_MAC\_GENERAL mechanism. It is defined as follows:

```
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:

638 *ulwordSize* wordsize of RC5 cipher in bytes

639 *ulRounds* number of rounds of RC5 encipherment

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

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

## 2.6.4 RC5 key generation

- The RC5 key generation mechanism, denoted **CKM RC5 KEY GEN**, is a key generation mechanism for
- 644 RSA Security's block cipher RC5.
- 645 It does not have a parameter.

642

- The mechanism generates RC5 keys with a particular length in bytes, as specified in the
- **CKA\_VALUE\_LEN** attribute of the template for the key.
- The mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and CKA\_VALUE attributes to the new
- key. Other attributes supported by the 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.

#### 653 **2.6.5 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
- 656 codebook mode as defined in FIPS PUB 81.
- 157 It has a parameter, **CK\_RC5\_PARAMS**, which indicates the wordsize and number of rounds of
- encryption to use.
- This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to
- wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 661 CKA VALUE attribute of the key that is wrapped, padded on the trailing end with null bytes so that the
- resulting length is a multiple of 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
- 666 CKA KEY TYPE attributes of the template and, if it has one, and the key type supports it, the
- 667 **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.
- 669 Constraints on key types and the length of data are summarized in the following table:
- 670 Table 15, RC5-ECB Key and Data Length

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

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

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

#### 2.6.6 RC5-CBC

673

693

- RC5-CBC, denoted **CKM\_RC5\_CBC**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and cipher-block chaining mode as defined in FIPS PUB 81.
- It has a parameter, a **CK\_RC5\_CBC\_PARAMS** structure, which specifies the wordsize and number of rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.
- This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the CKA\_VALUE attribute of the key that is wrapped, padded on the trailing end with up to seven null bytes so that the resulting length is a multiple of eight. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key; the application must convey these separately.
- For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the

  CKA\_KEY\_TYPE attribute for the template, and, if it has one, and the key type supports it, the

  CKA\_VALUE\_LEN attribute of the template. The mechanism contributes the result as the CKA\_VALUE

  attribute of the new key; other attributes required by the key type must be specified in the template.
- 689 Constraints on key types and the length of data are summarized in the following table:

## 690 Table 16, RC5-CBC Key and Data Length

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

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

# 2.6.7 RC5-CBC with PKCS padding

RC5-CBC with PKCS padding, denoted **CKM\_RC5\_CBC\_PAD**, is a mechanism for single- and 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.

- It has a parameter, a **CK\_RC5\_CBC\_PARAMS** structure, which specifies the wordsize and number of rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.
- The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
- 702 for the **CKA VALUE LEN** attribute.
- 703 In addition to being able to wrap an unwrap secret keys, this mechanism MAY wrap and unwrap RSA,
- 704 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys. The entries in
- the table below for data length constraints when wrapping and unwrapping keys do not apply to wrapping
- and unwrapping private keys.
- Constraints on key types and the length of data are summarized in the following table:
- 708 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.
- 711 2.6.8 General-length RC5-MAC
- 712 General-length RC5-MAC, denoted **CKM RC5 MAC GENERAL**, is a mechanism for single- and
- 713 multiple-part signatures and verification, based on RSA Security's block cipher RC5 and data
- authentication as defined in FIPS PUB 113.
- 715 It has a parameter, a CK RC5 MAC GENERAL PARAMS structure, which specifies the wordsize and
- 716 number of rounds of encryption to use and the output length desired from the mechanism.
- 717 The output bytes from this mechanism are taken from the start of the final RC5 cipher block produced in
- 718 the MACing process.
- Constraints on key types and the length of data are summarized in the following table:
- 720 Table 18, General-length RC2-MAC: Key and Data Length

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

- For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure
- specify the supported range of RC5 key sizes, in bytes.
- 723 **2.6.9 RC5-MAC**
- 724 RC5-MAC, denoted by **CKM\_RC5\_MAC**, is a special case of the general-length RC5-MAC mechanism.
- 725 Instead of taking a CK RC5 MAC GENERAL PARAMS parameter, it takes a CK RC5 PARAMS
- parameter. RC5-MAC produces and verifies MACs half as large as the RC5 blocksize.
- 727 Constraints on key types and the length of data are summarized in the following table:

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

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

# 2.7 General block cipher

#### 732 **2.7.1 Definitions**

- 733 For brevity's sake, the mechanisms for the DES, CAST, CAST3, CAST128, IDEA and CDMF block
- 734 ciphers are described together here. Each of these ciphers ha the following mechanisms, which are
- 735 described in a templatized form.
- This section defines the key types "CKK\_DES", "CKK\_CAST", "CKK\_CAST3", "CKK\_CAST128",
- "CKK\_IDEA" and "CKK\_CDMF" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key
- 738 objects.

731

739 Mechanisms:

```
740 CKM_DES_KEY_GEN
```

- 741 CKM\_DES\_ECB
- 742 CKM DES CBC
- 743 CKM DES MAC
- 744 CKM DES MAC GENERAL
- 745 CKM\_DES\_CBC\_PAD
- 746 CKM\_CDMF\_KEY\_GEN
- 747 CKM\_CDMF\_ECB
- 748 CKM\_CDMF\_CBC
- 749 CKM CDMF MAC
- 750 CKM\_CDMF\_MAC\_GENERAL
- 751 CKM\_CDMF\_CBC\_PAD
- 752 CKM\_DES\_OFB64
- 753 CKM DES OFB8
- 754 CKM\_DES\_CFB64
- 755 CKM DES CFB8
- 756 CKM CAST KEY GEN
- 757 CKM\_CAST\_ECB
- 758 CKM CAST CBC
- 759 CKM CAST MAC
- 760 CKM\_CAST\_MAC\_GENERAL
- 761 CKM CAST CBC PAD
- 762 CKM\_CAST3\_KEY\_GEN
- 763 CKM\_CAST3\_ECB
- 764 CKM\_CAST3\_CBC
- 765 CKM CAST3 MAC

```
766
           CKM CAST3 MAC GENERAL
767
           CKM_CAST3_CBC_PAD
768
           CKM_CAST128_KEY_GEN
           CKM CAST128 ECB
769
770
           CKM CAST128 CBC
771
           CKM CAST128 MAC
772
           CKM CAST128 MAC GENERAL
           CKM_CAST128_CBC_PAD
773
774
           CKM_IDEA_KEY_GEN
775
           CKM IDEA ECB
           CKM IDEA MAC
776
777
           CKM IDEA MAC GENERAL
778
           CKM_IDEA_CBC_PAD
```

## 779 2.7.2 DES secret key objects

- 780 DES secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_DES) hold single-length DES
- 781 keys. The following table defines the DES secret key object attributes, in addition to the common
- 782 attributes defined for this object class:
- 783 Table 20, DES Secret Key Object

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

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

```
788
           CK OBJECT CLASS class = CKO SECRET KEY;
789
          CK KEY TYPE keyType = CKK DES;
          CK UTF8CHAR label[] = "A DES secret key object";
790
791
          CK BYTE value[8] = {...};
792
          CK BBOOL true = CK TRUE;
793
          CK ATTRIBUTE template[] = {
794
             {CKA CLASS, &class, sizeof(class)},
795
             {CKA KEY TYPE, & keyType, sizeof(keyType)},
796
             {CKA TOKEN, &true, sizeof(true)},
797
             {CKA LABEL, label, sizeof(label)-1},
798
             {CKA ENCRYPT, &true, sizeof(true)},
799
             {CKA VALUE, value, sizeof(value}
800
          };
```

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.

## 2.7.3 CAST secret key objects

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

804

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

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

809 810 811

825

827

845

847

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

```
812
          CK OBJECT CLASS class = CKO SECRET KEY;
813
          CK KEY TYPE keyType = CKK CAST;
          CK UTF8CHAR label[] = "A CAST secret key object";
814
815
          CK BYTE value[] = {...};
816
          CK BBOOL true = CK TRUE;
817
          CK ATTRIBUTE template[] = {
818
              {CKA CLASS, &class, sizeof(class)},
819
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
820
             {CKA TOKEN, &true, sizeof(true)},
821
             {CKA LABEL, label, sizeof(label)-1},
822
             {CKA ENCRYPT, &true, sizeof(true)},
823
             {CKA VALUE, value, sizeof(value)}
824
```

## 2.7.4 CAST3 secret key objects

826 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:

829 Table 22, CAST3 Secret Key Object Attributes

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

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

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

```
832
          CK OBJECT CLASS class = CKO SECRET KEY;
833
          CK KEY TYPE keyType = CKK CAST3;
834
          CK_UTF8CHAR label[] = "A CAST3 secret key object";
835
          CK BYTE value[] = {...};
836
          CK BBOOL true = CK TRUE;
837
          CK ATTRIBUTE template[] = {
838
             {CKA CLASS, &class, sizeof(class)},
839
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
840
             {CKA TOKEN, &true, sizeof(true)},
841
             {CKA LABEL, label, sizeof(label)-1},
842
             {CKA ENCRYPT, &true, sizeof(true)},
843
             {CKA VALUE, value, sizeof(value)}
844
          };
```

## 2.7.5 CAST128 secret key objects

846 CAST128 secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_CAST128) hold

CAST128 keys. The following table defines the CAST128 secret key object attributes, in addition to the

common attributes defines for this object class:

851

865

869

870

884

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

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

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

```
852
          CK OBJECT CLASS class = CKO SECRET KEY;
853
          CK KEY TYPE keyType = CKK CAST128;
854
          CK_UTF8CHAR label[] = "A CAST128 secret key object";
855
          CK BYTE value[] = {...};
856
          CK BBOOL true = CK TRUE;
857
          CK ATTRIBUTE template[] = {
858
             {CKA CLASS, &class, sizeof(class)},
859
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
860
             {CKA TOKEN, &true, sizeof(true)},
861
             {CKA LABEL, label, sizeof(label)-1},
             {CKA ENCRYPT, &true, sizeof(true)},
862
863
             {CKA VALUE, value, sizeof(value)}
864
          };
```

## 2.7.6 IDEA secret key objects

866 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:

868 Table 24, IDEA Secret Key Object

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

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

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

```
871
          CK OBJECT CLASS class = CKO SECRET KEY;
872
          CK KEY TYPE keyType = CKK IDEA;
873
          CK UTF8CHAR label[] = "An IDEA secret key object";
874
          CK BYTE value [16] = \{...\};
875
          CK BBOOL true = CK_TRUE;
876
          CK ATTRIBUTE template[] = {
877
              {CKA_CLASS, &class, sizeof(class)},
              {CKA KEY_TYPE, &keyType, sizeof(keyType)},
878
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
```

## 2.7.7 CDMF secret key objects

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

887 Table 25, CDMF Secret Key Object

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

- Refer to [PKCS #11-Base] table 11 for footnotes
- 889 CDMF keys MUST have their parity bits properly set in exactly the same fashion described for DES keys
- in FIPS PUB 46-3. Attempting to create or unwrap a CDMF key with incorrect parity MUST return an
- 891 error.

906

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

```
893
          CK OBJECT CLASS class = CKO SECRET KEY;
894
          CK KEY TYPE keyType = CKK CDMF;
895
          CK UTF8CHAR label[] = "A CDMF secret key object";
896
          CK BYTE value[8] = {...};
          CK BBOOL true = CK TRUE;
897
898
          CK ATTRIBUTE template[] = {
899
              {CKA CLASS, &class, sizeof(class)},
900
             {CKA KEY TYPE, &keyType, sizeof(keyType)},
901
             {CKA TOKEN, &true, sizeof(true)},
902
             {CKA LABEL, label, sizeof(label)-1},
903
             {CKA ENCRYPT, &true, sizeof(true)},
904
             {CKA VALUE, value, sizeof(value)}
905
          };
```

## 2.7.8 General block cipher mechanism parameters

## 907 2.7.8.1 CK MAC GENERAL PARAMS; CK MAC GENERAL PARAMS PTR

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

```
913 typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
914
```

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

#### 916 2.7.9 General block cipher key generation

- 917 Cipher <NAME> has a key generation mechanism, "<NAME> key generation", denoted by
- 918 CKM\_<NAME>\_KEY\_GEN.
- 919 This mechanism does not have a parameter.
- 920 The mechanism contributes the CKA CLASS, CKA KEY TYPE, and CKA VALUE attributes to the new
- 921 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key
- 922 supports) MAY be specified in the template for the key, or else are assigned default initial values.
- 923 When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB
- 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits
- 925 set properly.
- 926 When DES or CDMF keys are generated, it is token-dependent whether or not it is possible for "weak" or
- 927 "semi-weak" keys to be generated. Similarly, when triple-DES keys are generated, it is token-dependent
- 928 whether or not it is possible for any of the component DES keys to be "weak" or "semi-weak" keys.
- 929 When CAST, CAST3, or CAST128 keys are generated, the template for the secret key must specify a
- 930 **CKA\_VALUE\_LEN** attribute.
- 931 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK MECHANISM INFO** structure
- 932 MAY be used. The CAST, CAST3, and CAST128 ciphers have variable key sizes, and so for the key
- 933 generation mechanisms for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the
- 934 **CK MECHANISM INFO** structure specify the supported range of key sizes, in bytes. For the DES,
- 935 DES3 (triple-DES), IDEA and CDMF ciphers, these fields and not used.

## 2.7.10 General block cipher ECB

- 937 Cipher <NAME> has an electronic codebook mechanism, "<NAME>-ECB", denoted
- 938 **CKM\_<NAME>\_ECB**. It is a mechanism for single- and multiple-part encryption and decryption; key
- 939 wrapping; and key unwrapping with <NAME>.
- 940 It does not have a parameter.

936

- This mechanism MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to
- 942 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the
- 943 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with null bytes so that the
- 944 resulting length is a multiple of <NAME>'s blocksize. The output data is the same length as the padded
- 945 input data. It does not wrap the key type, key length or any other information about the key; the
- application must convey these separately.
- 947 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the
- 948 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the
- 949 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**
- attribute of the new key; other attributes required by the key must be specified in the template.
- Constraints on key types and the length of data are summarized in the following table:
- 952 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	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySIze* fields of the **CK\_MECHANISM\_INFO** structure
MAY be used. The CAST, CAST3, and CAST128 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

957 fields are not used.

958

# 2.7.11 General block cipher CBC

- Cipher <NAME> has a cipher-block chaining mode, "<NAME>-CBC", denoted **CKM\_<NAME>\_CBC**. It is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping with <NAME>.
- 962 It has a parameter, an initialization vector for cipher block chaining mode. The initialization vector has the same length as <NAME>'s blocksize.
- 964 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

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

## 2.7.12 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.

976 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.

981

971

978

979

980

982

983

984

985 986

993

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

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

987 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 blocksize	Between 1 and blocksize bytes shorter than input length

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

# 2.7.13 General-length general block cipher MAC

Cipher <NAME> has a general-length MACing mode, "General-length <NAME>-MAC", denoted CKM\_<NAME>\_MAC\_GENERAL. It is a mechanism for single-and multiple-part signatures and

- 996 verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB
- 997 113.
- 998 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
- 1000 MACing process.
- 1001 Constraints on key types and the length of input and output data are summarized in the following table:
- 1002 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

- 1003 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK MECHANISM INFO** structure
- 1004 MAY be used. The CAST, CAST3, and CAST128 ciphers have variable key sizes, and so for these
- 1005 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
- 1007 fields are not used.

1008

1020

#### 2.7.14 General block cipher MAC

- 1009 Cipher <NAME> has a MACing mechanism, "<NAME>-MAC", denoted **CKM <NAME> MAC**. This
- 1010 mechanism is a special case of the CKM <NAME> MAC GENERAL mechanism described above. It
- 1011 produces an output of size half as large as <NAME>'s blocksize.
- 1012 This mechanism has no parameters.
- 1013 Constraints on key types and the length of data are summarized in the following table:
- 1014 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]

- 1015 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK MECHANISM INFO** structure
- 1016 MAY be used. The CAST, CAST3, and CAST128 ciphers have variable key sizes, and so for these
- 1017 ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the
- 1018 supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these
- 1019 fields are not used.

#### 2.8 SKIPJACK

#### **2.8.1 Definitions**

- 1022 This section defines the key type "CKK SKIPJACK" for type CK KEY TYPE as used in the
- 1023 CKA KEY TYPE attribute of key objects.
- 1024 Mechanisms:
- 1025 CKM SKIPJACK KEY GEN
- 1026 CKM SKIPJACK ECB64
- 1027 CKM SKIPJACK CBC64
- 1028 CKM SKIPJACK OFB64
- 1029 CKM SKIPJACK CFB64

```
1030 CKM_SKIPJACK_CFB32
1031 CKM_SKIPJACK_CFB16
1032 CKM_SKIPJACK_CFB8
1033 CKM_SKIPJACK_WRAP
1034 CKM_SKIPJACK_PRIVATE_WRAP
1035 CKM_SKIPJACK_RELAYX
```

### 2.8.2 SKIPJACK secret key objects

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

1040 Table 31, SKIPJACK Secret Key Object

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

1041 Refer to [PKCS #11-Base] table 11 for footnotes

1042

1045

1046

1047

1061

1036

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

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

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

```
1048
           CK OBJECT CLASS class = CKO SECRET KEY;
1049
           CK KEY TYPE keyType = CKK SKIPJACK;
           CK UTF8CHAR label[] = "A SKIPJACK MEK secret key object";
1050
1051
           CK BYTE value[12] = {...};
1052
           CK BBOOL true = CK TRUE;
1053
           CK ATTRIBUTE template[] = {
1054
              {CKA CLASS, &class, sizeof(class)},
1055
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1056
              {CKA TOKEN, &true, sizeof(true)},
1057
              {CKA LABEL, label, sizeof(label)-1},
1058
              {CKA ENCRYPT, &true, sizeof(true)},
              {CKA VALUE, value, sizeof(value)}
1059
1060
```

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

```
1062
           CK OBJECT CLASS class = CKO SECRET KEY;
1063
           CK KEY TYPE keyType = CKK SKIPJACK;
1064
           CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1065
           CK BYTE value[12] = \{...\};
1066
           CK BBOOL true = CK TRUE;
           CK ATTRIBUTE template[] = {
1067
1068
              {CKA CLASS, &class, sizeof(class)},
1069
              {CKA KEY TYPE, &keyType, sizeof(keyType)},
1070
              {CKA_TOKEN, &true, sizeof(true)},
1071
              {CKA LABEL, label, sizeof(label)-1},
1072
              {CKA ENCRYPT, &true, sizeof(true)},
1073
              {CKA WRAP, &true, sizeof(true)},
1074
              {CKA VALUE, value, sizeof(value)}
1075
           };
```

#### 2.8.3 SKIPJACK Mechanism parameters 1076 2.8.3.1 CK SKIPJACK PRIVATE WRAP PARAMS; 1077 1078 CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR 1079 CK SKIPJACK PRIVATE WRAP PARAMS is a structure that provides the parameters to the 1080 CKM SKIPJACK PRIVATE WRAP mechanism. It is defined as follows: 1081 typedef struct CK SKIPJACK PRIVATE WRAP PARAMS { 1082 CK ULONG ulPasswordLen; 1083 CK BYTE PTR pPassword; 1084 CK ULONG ulPublicDataLen; 1085 CK BYTE PTR pPublicData; 1086 CK ULONG ulPandGLen; 1087 CK ULONG ulQLen; 1088 CK ULONG ulRandomLen; 1089 CK BYTE PTR pRandomA; 1090 CK BYTE PTR pPrimeP; 1091 CK BYTE PTR pBaseG; 1092 CK BYTE PTR pSubprimeQ; 1093 } CK SKIPJACK PRIVATE WRAP PARAMS; 1094 The fields of the structure have the following meanings: 1095 ulPasswordLen length of the password pointer to the buffer which contains the user-supplied 1096 pPassword password 1097 ulPublicDataLen other party's key exchange public key size 1098 1099 pPublicData pointer to other party's key exchange public key value ulPandGLen length of prime and base values 1100 ulQLen length of subprime value 1101 ulRandomLen size of random Ra, in bytes 1102 1103 *p*PrimeP pointer to Prime, p, value pointer to Base, b, value 1104 pBaseG pSubprimeQ pointer to Subprime, q, value 1105 CK SKIPJACK PRIVATE WRAP PARAMS PTR is a pointer to a 1106 CK PRIVATE WRAP PARAMS. 1107 1108 2.8.3.2 CK SKIPJACK RELAYX PARAMS; CK SKIPJACK RELAYX PARAMS PTR 1109 1110 CK\_SKIPJACK\_RELAYX\_PARAMS is a structure that provides the parameters to the 1111 CKM SKIPJACK RELAYX mechanism. It is defined as follows:

1112 1113

1114

typedef struct CK SKIPJACK RELAYX PARAMS {

CK ULONG uloldWrappedXLen;

CK BYTE PTR pOldWrappedX;

```
1115
              CK ULONG ulOldPasswordLen;
1116
              CK BYTE PTR pOldPassword;
1117
              CK ULONG ulOldPublicDataLen;
1118
             CK BYTE PTR pOldPublicData;
1119
             CK ULONG ulOldRandomLen;
1120
              CK BYTE PTR pOldRandomA;
1121
              CK ULONG ulNewPasswordLen;
1122
              CK BYTE PTR pNewPassword;
1123
              CK ULONG ulNewPublicDataLen;
1124
              CK BYTE PTR pNewPublicData;
1125
              CK ULONG ulNewRandomLen;
1126
              CK BYTE PTR pNewRandomA;
1127
             CK SKIPJACK RELAYX PARAMS;
1128
       The fields of the structure have the following meanings:
1129
                                      length of old wrapped key in bytes
                ulOldWrappedLen
                   pOldWrappedX
                                      pointer to old wrapper key
1130
               ulOldPasswordLen
                                      length of the old password
1131
1132
                   pOldPassword
                                      pointer to the buffer which contains the old user-supplied
1133
                                      password
              ulOldPublicDataLen
1134
                                      old key exchange public key size
1135
                  pOldPublicData
                                      pointer to old key exchange public key value
                 ulOldRandomLen
                                      size of old random Ra in bytes
1136
                   pOldRandomA
                                      pointer to old Ra data
1137
              ulNewPasswordLen
1138
                                      length of the new password
                  pNewPassword
                                      pointer to the buffer which contains the new user-
1139
1140
                                      supplied password
1141
             ulNewPublicDataLen
                                      new key exchange public key size
                 pNewPublicData
                                      pointer to new key exchange public key value
1142
                ulNewRandomLen
                                      size of new random Ra in bytes
1143
                                      pointer to new Ra data
1144
                  pNewRandomA
1145
       CK_SKIPJACK_RELAYX_PARAMS_PTR is a pointer to a CK_SKIPJACK_RELAYX_PARAMS.
1146
       2.8.4 SKIPJACK key generation
1147
       The SKIPJACK key generation mechanism, denoted CKM_SKIPJACK_KEY_GEN, is a key generation
1148
       mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).
1149
       It does not have a parameter.
1150
       The mechanism contributes the CKA_CLASS, CKA_KEY_TYPE, and CKA_VALUE attributes to the new
1151
       key.
```

#### 1152 **2.8.5 SKIPJACK-ECB64**

- 1153 SKIPJACK-ECB64, denoted **CKM\_SKIPJACK\_ECB64**, is a mechanism for single- and multiple-part
- encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB
- 1155 185

1161

- 1156 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1157 value generated by the token in other words, the application cant specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1159 Constraints on key types and the length of data are summarized in the following table:
- 1160 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

#### 2.8.6 SKIPJACK-CBC64

- 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.
- 1164 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1167 Constraints on key types and the length of data are summarized in the following table:
- 1168 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

#### 1169 **2.8.7 SKIPJACK-OFB64**

- 1170 SKIPJACK-OFB64, denoted **CKM SKIPJACK OFB64**, is a mechanism for single- and multiple-part
- 1171 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.
- 1172 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1175 Constraints on key types and the length of data are summarized in the following table:
- 1176 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

#### 1177 **2.8.8 SKIPJACK-CFB64**

- 1178 SKIPJACK-CFB64, denoted **CKM\_SKIPJACK\_CFB64**, is a mechanism for single- and multiple-part
- 1179 encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.

- 1180 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1181 value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1183 Constraints on key types and the length of data are summarized in the following table:
- 1184 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

#### 1185 **2.8.9 SKIPJACK-CFB32**

- SKIPJACK-CFB32, denoted **CKM\_SKIPJACK\_CFB32**, is a mechanism for single- and multiple-part
- encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.
- 1188 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1191 Constraints on key types and the length of data are summarized in the following table:
- 1192 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

#### 1193 **2.8.10 SKIPJACK-CFB16**

- SKIPJACK-CFB16, denoted **CKM\_SKIPJACK\_CFB16**, is a mechanism for single- and multiple-part
- encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.
- 1196 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1197 value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1199 Constraints on key types and the length of data are summarized in the following table:
- 1200 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

#### 1201 **2.8.11 SKIPJACK-CFB8**

- SKIPJACK-CFB8, denoted **CKM\_SKIPJACK\_CFB8**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.
- 1204 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1205 value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1207 Constraints on key types and the length of data are summarized in the following table:

1208 Table 38, SKIPJACK-CFB8: Data and Length

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

#### 2.8.12 SKIPJACK-WRAP 1209

- 1210 The SKIPJACK-WRAP mechanism, denoted CKM SKIPJACK WRAP, is used to wrap and unwrap a
- 1211 secret key (MEK). It MAY wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.
- 1212 It does not have a parameter.

#### 2.8.13 SKIPJACK-PRIVATE-WRAP 1213

- 1214 The SKIPJACK-PRIVATE-WRAP mechanism, denoted CKM SKIPJACK PRIVATE WRAP, is used to
- 1215 wrap and unwrap a private key. It MAY wrap KEA and DSA private keys.
- It has a parameter, a **CK SKIPJACK PRIVATE WRAP PARAMS** structure. 1216

#### 1217 2.8.14 SKIPJACK-RELAYX

- 1218 The SKIPJACK-RELAYX mechanism, denoted CKM\_SKIPJACK\_RELAYX, is used with the C\_WrapKey
- 1219 function to "change the wrapping" on a private key which was wrapped with the SKIPJACK-PRIVATE-
- WRAP mechanism (See Section 2.8.13). 1220
- 1221 It has a parameter, a CK SKIPJACK RELAYX PARAMS structure.
- 1222 Although the SKIPJACK-RELAYX mechanism is used with C\_WrapKey, it differs from other key-
- wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to 1223
- C WrapKey; however for the SKIPJACK RELAYX mechanism, the [always invalid] value 0 should be 1224
- passed as the key handle for **C WrapKey**, and the already-wrapped key should be passed in as part of 1225
- 1226 the CK SKIPJACK RELAYX PARAMS structure.

#### **2.9 BATON** 1227

#### 2.9.1 Definitions 1228

- 1229 This section defines the key type "CKK BATON" for type CK KEY TYPE as used in the
- CKA KEY TYPE attribute of key objects. 1230
- 1231 Mechanisms:

1237

- 1232 CKM BATON KEY GEN
- 1233 CKM\_BATON\_ECB128
- 1234 CKM BATON ECB96
- 1235 CKM\_BATON\_CBC128
- 1236 CKM\_BATON\_COUNTER
- 1238 CKM BATON WRAP

#### 2.9.2 BATON secret key objects 1239

CKM BATON SHUFFLE

- 1240 BATON secret key objects (object class CKO SECRET KEY, key type CKK BATON) hold single-length
- 1241 BATON keys. The following table defines the BATON secret key object attributes, in addition to the
- 1242 common attributes defined for this object class:

#### Attribute

#### Data type Meaning

CKA\_VALUE<sup>1,4,6,7</sup> Byte array Key value (40 bytes long)

1244 Refer to [PKCS #11-Base] table 11 for footnotes

1245

1264

- 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 MUST return an error.
- 1248 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key 1249 with a specified value. Nonetheless, we provide templates for doing so.
- 1250 The following is a sample template for creating a BATON MEK secret key object:

```
1251
            CK OBJECT CLASS class = CKO SECRET KEY;
1252
            CK KEY TYPE keyType = CKK BATON;
            CK_UTF8CHAR label[] = "A BATON MEK secret key object";
1253
1254
            CK BYTE value[40] = {...};
1255
            CK BBOOL true = CK TRUE;
1256
            CK ATTRIBUTE template[] = {
1257
               {CKA CLASS, &class, sizeof(class)},
1258
               {CKA KEY TYPE, &keyType, sizeof(keyType)},
1259
               {CKA TOKEN, &true, sizeof(true)},
1260
              {CKA LABEL, label, sizeof(label)-1},
1261
              {CKA ENCRYPT, &true, sizeof(true)},
1262
              {CKA VALUE, value, sizeof(value)}
1263
            };
```

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

```
1265
            CK OBJECT CLASS class = CKO SECRET KEY;
1266
            CK KEY TYPE keyType = CKK BATON;
1267
            CK UTF8CHAR label[] = "A BATON TEK secret key object";
1268
            CK BYTE value[40] = {...};
1269
            CK BBOOL true = CK TRUE;
1270
            CK ATTRIBUTE template[] = {
1271
              {CKA CLASS, &class, sizeof(class)},
1272
              {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1273
              {CKA_TOKEN, &true, sizeof(true)},
1274
              {CKA LABEL, label, sizeof(label)-1},
1275
              {CKA ENCRYPT, &true, sizeof(true)},
1276
              {CKA WRAP, &true, sizeof(true)},
1277
              {CKA VALUE, value, sizeof(value)}
1278
            };
```

### 1279 **2.9.3 BATON key generation**

- 1280 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).
- 1282 It does not have a parameter.
- 1283 The mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and CKA\_VALUE attributes to the new
- 1284 key.

1285

#### 2.9.4 BATON-ECB128

- 1286 BATON-ECB128, denoted CKM BATON ECB128, is a mechanism for single- and multiple-part
- encryption and decryption with BATON in 128-bit electronic codebook mode.

- 1288 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1291 Constraints on key types and the length of data are summarized in the following table:
- 1292 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

#### 1293 **2.9.5 BATON-ECB96**

- BATON-ECB96, denoted **CKM\_BATON\_ECB96**, is a mechanism for single- and multiple-part encryption
- and decryption with BATON in 96-bit electronic codebook mode.
- 1296 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1299 Constraints on key types and the length of data are summarized in the following table:
- 1300 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

#### 1301 **2.9.6 BATON-CBC128**

- 1302 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.
- 1304 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1305 value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1307 Constraints on key types and the length of data are summarized in the following table:
- 1308 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

#### 1309 **2.9.7 BATON-COUNTER**

- BATON-COUNTER, denoted **CKM\_BATON\_COUNTER**, is a mechanism for single- and multiple-part encryption and decryption with BATON in counter mode.
- 1312 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1315 Constraints on key types and the length of data are summarized in the following table:

#### 1316 Table 43, BATON-COUNTER: Data and Length

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

#### **2.9.8 BATON-SHUFFLE**

- 1318 BATON-SHUFFLE, denoted **CKM\_BATON\_SHUFFLE**, is a mechanism for single- and multiple-part
- encryption and decryption with BATON in shuffle mode.
- 1320 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when
- encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1323 Constraints on key types and the length of data are summarized in the following table:
- 1324 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

#### 1325 **2.9.9 BATON WRAP**

- 1326 The BATON wrap and unwrap mechanism, denoted **CKM\_BATON\_WRAP**, is a function used to wrap
- 1327 and unwrap a secret key (MEK). It MAY wrap and unwrap SKIPJACK, BATON and JUNIPER keys.
- 1328 It has no parameters.
- 1329 When used to unwrap a key, this mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and
- 1330 **CKA VALUE** attributes to it.

#### 1331 **2.10 JUNIPER**

#### **2.10.1 Definitions**

- 1333 This section defines the key type "CKK\_JUNIPER" for type CK\_KEY\_TYPE as used in the
- 1334 CKA KEY TYPE attribute of key objects.
- 1335 Mechanisms:

1342

- 1336 CKM\_JUNIPER\_KEY\_GEN
- 1337 CKM\_JUNIPER\_ECB128
- 1338 CKM JUNIPER CBC128
- 1339 CKM JUNIPER COUNTER
- 1340 CKM JUNIPER SHUFFLE
- 1341 CKM\_JUNIPER\_WRAP

#### 2.10.2 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:

#### Attribute

#### Data type Meaning

CKA VALUE<sup>1,4,6,7</sup> Byte array Key value (40 bytes long)

1347 Refer to [PKCS #11-Base] table 11 for footnotes

1348

1367

- 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 MUST return an error.
- 1351 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.
- 1353 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1354
            CK OBJECT CLASS class = CKO SECRET KEY;
1355
            CK KEY TYPE keyType = CKK JUNIPER;
            CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";
1356
1357
            CK BYTE value[40] = {...};
1358
            CK BBOOL true = CK TRUE;
1359
            CK ATTRIBUTE template[] = {
1360
               {CKA CLASS, &class, sizeof(class)},
1361
               {CKA KEY TYPE, &keyType, sizeof(keyType)},
1362
               {CKA TOKEN, &true, sizeof(true)},
1363
              {CKA LABEL, label, sizeof(label)-1},
1364
              {CKA ENCRYPT, &true, sizeof(true)},
1365
              {CKA VALUE, value, sizeof(value)}
1366
            };
```

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

```
1368
            CK OBJECT CLASS class = CKO SECRET KEY;
1369
            CK KEY TYPE keyType = CKK JUNIPER;
1370
            CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";
1371
            CK BYTE value[40] = {...};
1372
            CK BBOOL true = CK TRUE;
1373
            CK ATTRIBUTE template[] = {
1374
              {CKA CLASS, &class, sizeof(class)},
1375
              {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1376
              {CKA_TOKEN, &true, sizeof(true)},
1377
              {CKA LABEL, label, sizeof(label)-1},
1378
              {CKA ENCRYPT, &true, sizeof(true)},
1379
              {CKA WRAP, &true, sizeof(true)},
1380
              {CKA VALUE, value, sizeof(value)}
1381
            };
```

### 2.10.3 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).
- 1385 It does not have a parameter.
- The mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and CKA\_VALUE attributes to the new
- 1387 key.

1382

1388

#### 2.10.4 JUNIPER-ECB128

- 1389 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.

- 1391 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1392 value generated by the token – in other words, the application MAY NOT specify a particular IV when
- 1393 encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1394 Constraints on key types and the length of data are summarized in the following table. For encryption
- 1395 and decryption, the input and output data (parts) MAY begin at the same location in memory.
- 1396 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

#### 1397 2.10.5 JUNIPER-CBC128

- 1398 JUNIPER-CBC128, denoted CKM JUNIPER CBC128, is a mechanism for single- and multiple-part encryption and decryption with JUNIPER in 128-bit cipher block chaining mode. 1399
- 1400 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- 1401 value generated by the token – in other words, the application MAY NOT specify a particular IV when
- 1402 encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1403 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. 1404
- 1405 Table 47. JUNIPER-CBC128: Data and Length

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

#### 2.10.6 JUNIPER-COUNTER

1406

1415

- 1407 JUNIPER-COUNTER, denoted **CKM JUNIPER COUNTER**, is a mechanism for single- and multiplepart encryption and decryption with JUNIPER in counter mode. 1408
- 1409 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when 1410
- 1411 encrypting. It MAY, of course, specify a particular IV when decrypting.
- 1412 Constraints on key types and the length of data are summarized in the following table. For encryption
- 1413 and decryption, the input and output data (parts) MAY begin at the same location in memory.
- 1414 Table 48, JUNIPER-COUNTER: Data and Length

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

#### 2.10.7 JUNIPER-SHUFFLE

- 1416 JUNIPER-SHUFFLE, denoted CKM\_JUNIPER\_SHUFFLE, is a mechanism for single- and multiple-part 1417 encryption and decryption with JUNIPER in shuffle mode.
- 1418 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
- value generated by the token in other words, the application MAY NOT specify a particular IV when 1419
- encrypting. It MAY, of course, specify a particular IV when decrypting. 1420

1421	Constraints on key type	s 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.

1423 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

#### **1424 2.10.8 JUNIPER WRAP**

- The JUNIPER wrap and unwrap mechanism, denoted **CKM\_JUNIPER\_WRAP**, is a function used to wrap
- 1426 and unwrap an MEK. It MAY wrap or unwrap SKIPJACK, BATON and JUNIPER keys.
- 1427 It has no parameters.
- 1428 When used to unwrap a key, this mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and
- 1429 **CKA\_VALUE** attributes to it.
- 1430 **2.11 MD2**
- **2.11.1 Definitions**
- 1432 Mechanisms:
- 1433 CKM MD2
- 1434 CKM\_MD2\_HMAC
- 1435 CKM MD2 HMAC GENERAL
- 1436 CKM\_MD2\_KEY\_DERIVATION
- 1437 **2.11.2 MD2 digest**
- 1438 The MD2 mechanism, denoted **CKM\_MD2**, is a mechanism for message digesting, following the MD2
- 1439 message-digest algorithm defined in RFC 6149.
- 1440 It does not have a parameter.
- 1441 Constraints on the length of data are summarized in the following table:
- 1442 Table 50, MD2: Data Length

#### **Function Data length Digest Length**

C\_Digest Any 16

#### 1443 2.11.3 General-length MD2-HMAC

- The general-length MD2-HMAC mechanism, denoted **CKM\_MD2\_HMAC\_GENERAL**, is a mechanism for
- 1445 signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it
- 1446 uses are generic secret keys.
- 1447 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)
- 1449 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.
- 1450 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

#### 1451 **2.11.4 MD2-HMAC**

- 1452 The MD2-HMAC mechanism, denoted **CKM MD2 HMAC**, is a special case of the general-length MD2-
- 1453 HMAC mechanism in Section 2.11.3.
- 1454 It has no parameter, and produces an output of length 16.

## **2.11.5 MD2 key derivation**

- 1456 MD2 key derivation, denoted **CKM\_MD2\_KEY\_DERIVATION**, is a mechanism which provides the
- capability of deriving a secret key by digesting the value of another secret key with MD2.
- The value of the base key is digested once, and the result is used to make the value of the derived secret key.
- If no length or key type is provided in the template, then the key produced by this mechanism MUST be a generic secret key. Its length MUST be 16 bytes (the output size of MD2)..
- If no key type is provided in the template, but a length is, then the key produced by this mechanism MUST be a generic secret key of the specified length.
- If no length was provided in the template, but a key type is, then that key type must have a well-defined length. If it does, then the key produced by this mechanism MUST be of the type specified in the template. If it doesn't, an error MUST be returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by this mechanism MUST be of the specified type and length.
- 1469 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set 1470 properly.
- 1471 If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.
- 1472 This mechanism has the following rules about key sensitivity and extractability:
- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its CKA\_ALWAYS\_SENSITIVE attribute set to CK\_FALSE, then the derived key
  MUST as well. If the base key has its CKA\_ALWAYS\_SENSITIVE attribute set to CK\_TRUE, then
  the derived key has its CKA\_ALWAYS\_SENSITIVE attribute set to the same value as its

  CKA\_SENSITIVE attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_FALSE**, then the derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_TRUE**, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

#### 1484 **2.12 MD5**

#### **2.12.1 Definitions**

1486 Mechanisms:

1487 CKM MD5

1488 CKM MD5 HMAC

1489 CKM\_MD5\_HMAC\_GENERAL 1490 CKM MD5 KEY DERIVATION

#### 1491 **2.12.2 MD5 Digest**

- The MD5 mechanism, denoted **CKM\_MD5**, is a mechanism for message digesting, following the MD5
- 1493 message-digest algorithm defined in RFC 1321.
- 1494 It does not have a parameter.
- 1495 Constraints on the length of input and output data are summarized in the following table. For single-part
- digesting, the data and the digest MAY begin at the same location in memory.
- 1497 Table 52, MD5: Data Length

Function	Data length	Digest length
C_Digest	Any	16

#### 1498 2.12.3 General-length MD5-HMAC

- 1499 The general-length MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC\_GENERAL**, is a mechanism for
- 1500 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it
- uses are generic secret keys.
- 1502 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)
- produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.
- 1505 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

#### 1506 **2.12.4 MD5-HMAC**

- The MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC**, is a special case of the general-length MD5-
- 1508 HMAC mechanism in Section 2.12.3.
- 1509 It has no parameter, and produces an output of length 16.

#### **2.12.5 MD5 key derivation**

- 1511 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.
- 1513 The value of the base key is digested once, and the result is used to make the value of derived secret
- 1514 key.
- If no length or key type is provided in the template, then the key produced by this mechanism MUST be a generic secret key. Its length MUST be 16 bytes (the output size of MD5).
- If no key type is provided in the template, but a length is, then the key produced by this mechanism MUST be a generic secret key of the specified length.
- If no length was provided in the template, but a key type is, then that key type must have a welldefined length. If it does, then the key produced by this mechanism MUST be of the type specified in the template. If it doesn't, an error MUST be returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by this mechanism MUST be of the specified type and length.
- 1524 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set properly.
- 1526 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.

- 1527 This mechanism has the following rules about key sensitivity and extractability.
- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY both be specified to 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
  1532 MUST as well. If the base key has its CKA\_ALWAYS\_SENSITIVE attribute set to CK\_TRUE, then
  1533 the derived key has its CKA\_ALWAYS\_SENSITIVE attribute set to the same value as its
  1534 CKA SENSITIVE attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_FALSE**, then the derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_TRUE**, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

#### 1539 **2.13 FASTHASH**

#### 1540 **2.13.1 Definitions**

- 1541 Mechanisms:
- 1542 CKM FASTHASH

#### 1543 2.13.2 FASTHASH digest

- The FASTHASH mechanism, denoted **CKM\_FASTHASH**, is a mechanism for message digesting,
- 1545 following the U.S. government's algorithm.
- 1546 It does not have a parameter.
- 1547 Constraints on the length of input and output data are summarized in the following table:
- 1548 Table 54, FASTHASH: Data Length

# Function Input length Digest length C Digest Any 40

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

#### 1550 **2.14.1 Definitions**

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

```
1554
            CKM_PBE_MD2_DES_CBC
1555
            CKM PBE MD5 DES CBC
1556
            CKM PBE MD5 CAST CBC
1557
            CKM PBE MD5 CAST3 CBC
1558
            CKM_PBE_MD5_CAST128_CBC
1559
            CKM_PBE_SHA1_CAST128_CBC
1560
            CKM PBE SHA1 RC4 128
1561
            CKM PBE SHA1 RC4 40
1562
            CKM PBE SHA1 RC2 128 CBC
1563
            CKM PBE SHA1 RC2 40 CBC
```

## 2.14.2 Password-based encryption/authentication mechanism parameters

#### 1565 2.14.2.1 CK\_PBE\_PARAMS; CK\_PBE\_PARAMS\_PTR

1564

1577

1586

1594

1566 **CK\_PBE\_PARAMS** is a structure which provides all of the necessary information required by the
1567 CKM\_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation
1568 mechanisms) and the CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC mechanism. It is defined as follows:

```
1569
            typedef struct CK PBE PARAMS {
1570
              CK_BYTE_PTR pInitVector;
              CK UTF8CHAR PTR pPassword;
1571
              CK ULONG ulPasswordLen;
1572
1573
              CK BYTE PTR pSalt;
1574
              CK ULONG ulSaltLen;
1575
              CK ULONG ulIteration;
1576
            } CK PBE PARAMS;
```

The fields of the structure have the following meanings:

1578 1579	plnitVector	pointer to the location that receives the 8-byte initialization vector (IV), if an IV is required
1580 1581	pPassword	points to the password to be used in the PBE key generation
1582	ulPasswordLen	length in bytes of the password information
1583	pSalt	points to the salt to be used in the PBE key generation
1584	ulSaltLen	length in bytes of the salt information
1585	ullteration	number of iterations required for the generation

#### CK\_PBE\_PARAMS\_PTR is a pointer to a CK\_PBE\_PARAMS.

#### 1587 **2.14.3 MD2-PBE for DES-CBC**

- MD2-PBE for DES-CBC, denoted **CKM\_PBE\_MD2\_DES\_CBC**, is a mechanism used for generating a DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an
- iteration count. This functionality is defined in PKCS #5 as PBKDF1.
- 1591 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 receives the 8-byte IV
- 1593 generated by the mechanism.

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

- 1595 MD5-PBE for DES-CBC, denoted **CKM PBE MD5 DES CBC**, is a mechanism used for generating a
- 1596 DES 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 defined in PKCS #5 as PBKDF1.
- 1598 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the
- 1599 key generation process and the location of the application-supplied buffer which receives the 8-byte IV
- 1600 generated by the mechanism.

#### 1601 **2.14.5 MD5-PBE for CAST-CBC**

- MD5-PBE for CAST-CBC, denoted CKM\_PBE\_MD5\_CAST\_CBC, is a mechanism used for generating a
- 1603 CAST 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.
- 1605 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 receives the 8-byte IV
- 1607 generated by the mechanism
- 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.

## 1610 **2.14.6 MD5-PBE for CAST3-CBC**

- 1611 MD5-PBE for CAST3-CBC, denoted CKM\_PBE\_MD5\_CAST3\_CBC, is a mechanism used for generating
- 1612 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.
- 1614 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 receives the 8-byte IV
- 1616 generated by the mechanism
- 1617 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.

#### 1619 **2.14.7 MD5-PBE for CAST128-CBC**

- 1620 MD5-PBE for CAST128-CBC, denoted **CKM\_PBE\_MD5\_CAST128\_CBC**, is a mechanism used for
- 1621 generating a CAST128 secret key and an IV from a password and a salt value by using the MD5 digest
- 1622 algorithm and an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for
- 1623 MD5 and DES.
- 1624 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 receives the 8-byte IV
- generated by the mechanism
- 1627 The length of the CAST128 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.

#### 1629 2.14.8 SHA-1-PBE for CAST128-CBC

- 1630 SHA-1-PBE for CAST128-CBC, denoted CKM PBE SHA1 CAST128, is a mechanism used for
- 1631 generating a CAST128 secret key and an IV from a password and salt value using the SHA-1 digest
- algorithm and an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for
- 1633 MD5 and DES.
- 1634 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 receives the 8-byte IV
- 1636 generated by the mechanism
- The length of the CAST128 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

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

#### 1640 **mechanisms**

#### 1641 **2.15.1 Definitions**

- The mechanisms in this section are for generating keys and IVs for performing password-based
- encryption or authentication. The method used to generate keys and IVs is based on a method that was
- 1644 specified in PKCS #12.

- We specify here a general method for producing various types of pseudo-random bits from a password,
- p; a string of salt bits, s; and an iteration count, c. The "type" of pseudo-random bits to be produced is
- identified by an identification byte, *ID*, described at the end of this section.
- Let H be a hash function built around a compression function  $\int : \mathbf{Z}_{2^{u}} \times \mathbf{Z}_{2^{v}} \to \mathbf{Z}_{2^{u}}$  (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
- 1650 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.
- 16. Construct a string, *D* (the "diversifier"), by concatenating *v*/8 copies of *ID*.
  - 2. Concatenate copies of the salt together to create a string S of length  $v \cdot \lceil s/v \rceil$  bits (the final copy of the salt MAY be truncated to create S). Note that if the salt is the empty string, then so is S
  - 3. Concatenate copies of the password together to create a string P of length  $v \cdot |p/v|$  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.
- 1660 5. Set j = |n/u|.

1654

1655

1656

1657 1658

1659

1661

1662

1663

1664

1665 1666

1667

1668

1669 1670

1679

- 6. For i=1, 2, ..., j, do the following:
  - a. Set  $A_i = H_c(D||I)$ , the  $c_{th}$  hash of D||I|. That is, compute the hash of D||I|; compute the hash of that hash; etc.; continue in this fashion until a total of c hashes have been computed, each on the result of the previous hash.
  - b. Concatenate copies of *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 I0, I1, ..., 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, ..., Ai together to form a pseudo-random bit string, A.
- 1671 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
- 1675 the value 2.
- 1676 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
- 1678 value 3.

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

- 1680 SHA-1-PBE for 128-bit RC4, denoted **CKM\_PBE\_SHA1\_RC4\_128**, is a mechanism used for generating
- 1681 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.
- 1683 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the
- key generation process. The parameter also has a field to hold the location of an application-supplied
- buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not
- 1686 require an IV.
- 1687 The key produced by this mechanism will typically be used for performing password-based encryption.

1688	2.15.3	SHA-1	<b>PBE</b>	for	40-	bit	RC4	4
------	--------	-------	------------	-----	-----	-----	-----	---

- SHA-1-PBE for 40-bit RC4, denoted CKM\_PBE\_SHA1\_RC4\_40, is a mechanism used for generating a
- 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
- iteration count. The method used to generate the key is described above.
- 1692 It has a parameter, a CK PBE PARAMS structure. The parameter specifies the input information for the
- 1693 key generation process. The parameter also has a field to hold the location of an application-supplied
- buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not
- 1695 require an IV.
- 1696 The key produced by this mechanism will typically be used for performing password-based encryption.

#### 1697 2.15.4 SHA-1\_PBE for 128-bit RC2-CBC

- SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_128\_CBC**, is a mechanism used for
- 1699 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest
- algorithm and an iteration count. The method used to generate the key and IV is described above.
- 1701 It has a parameter, a CK PBE PARAMS structure. The parameter specifies the input information for the
- key generation process and the location of an application-supplied buffer which receives the 8-byte IV
- 1703 generated by the mechanism.
- 1704 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
- of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object
- 1706 Identifier pbeWithSHA1And128BitRC2-CBC.
- 1707 The key and IV produced by this mechanism will typically be used for performing password-based
- 1708 encryption.

#### 1709 2.15.5 SHA-1 PBE for 40-bit RC2-CBC

- 1710 SHA-1-PBE for 40-bit RC2-CBC, denoted CKM\_PBE\_SHA1\_RC2\_40\_CBC, is a mechanism used for
- 1711 generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm
- and an iteration count. The method used to generate the key and IV is described above.
- 1713 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the
- 1714 key generation process and the location of an application-supplied buffer which receives the 8-byte IV
- 1715 generated by the mechanism.
- 1716 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
- 1717 of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object
- 1718 Identifier pbeWithSHA1And40BitRC2-CBC.
- 1719 The key and IV produced by this mechanism will typically be used for performing password-based
- 1720 encryption

#### 1721 **2.16 RIPE-MD**

#### 1722 **2.16.1 Definitions**

- 1723 Mechanisms:
- 1724 CKM RIPEMD128
- 1725 CKM RIPEMD128 HMAC
- 1726 CKM RIPEMD128 HMAC GENERAL
- 1727 CKM\_RIPEMD160
- 1728 CKM RIPEMD160 HMAC
- 1729 CKM RIPEMD160 HMAC GENERAL

#### 1730 2.16.2 RIPE-MD 128 Digest

- 1731 The RIPE-MD 128 mechanism, denoted **CKM\_RIPEMD128**, is a mechanism for message digesting,
- 1732 following the RIPE-MD 128 message-digest algorithm.
- 1733 It does not have a parameter.
- 1734 Constraints on the length of data are summarized in the following table:
- 1735 Table 55, RIPE-MD 128: Data Length

## Function Data length Digest length

C Digest Any 16

1736

1737

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

- 1738 The general-length RIPE-MD 128-HMAC mechanism, denoted CKM\_RIPEMD128\_HMAC\_GENERAL, is
- 1739 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.
- 1741 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
- 1743 (MACs) produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.
- 1744 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

#### 1745 **2.16.4 RIPE-MD 128-HMAC**

- 1746 The RIPE-MD 128-HMAC mechanism, denoted **CKM RIPEMD128 HMAC**, is a special case of the
- 1747 general-length RIPE-MD 128-HMAC mechanism in Section 2.16.3.
- 1748 It has no parameter, and produces an output of length 16.

#### 1749 **2.16.5 RIPE-MD 160**

- 1750 The RIPE-MD 160 mechanism, denoted **CKM RIPEMD160**, is a mechanism for message digesting,
- following the RIPE-MD 160 message-digest defined in ISO-10118.
- 1752 It does not have a parameter.
- 1753 Constraints on the length of data are summarized in the following table:
- 1754 Table 57, RIPE-MD 160: Data Length

## Function Data length Digest length

C\_Digest Any 20

#### 1755 2.16.6 General-length RIPE-MD 160-HMAC

- 1756 The general-length RIPE-MD 160-HMAC mechanism, denoted **CKM RIPEMD160 HMAC GENERAL**, is
- 1757 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160
- 1758 hash function. The keys it uses are generic secret keys.

- 1759 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired
- output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures
- 1761 (MACs) produced by this mechanism MUST be taken from the start of the full 20-byte HMAC output.
- 1762 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

- 1763 **2.16.7 RIPE-MD 160-HMAC**
- 1764 The RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC**, is a special case of the
- general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.
- 1766 It has no parameter, and produces an output of length 20.
- 1767 **2.17 SET**
- 1768 **2.17.1 Definitions**
- 1769 Mechanisms:
- 1770 CKM\_KEY\_WRAP\_SET\_OAEP
- 1771 **2.17.2 SET mechanism parameters**
- 1772 2.17.2.1 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS; 1773 CK KEY WRAP SET OAEP PARAMS PTR
- 1774 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** is a structure that provides the parameters to the
- 1775 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;
```

- 1781 The fields of the structure have the following meanings:
- 1782 block contents byte
- 1783 pX concatenation of hash of plaintext data (if present) and
- 1784 extra data (if present)
- 1785 *ulXLen* length in bytes of concatenation of hash of plaintext data
- 1786 (if present) and extra data (if present). 0 if neither is
- 1787 present.
- 1788 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR is a pointer to a
- 1789 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS.
- 1790 2.17.3 OAEP key wrapping for SET
- The OAEP key wrapping for SET mechanism, denoted **CKM\_KEY\_WRAP\_SET\_OAEP**, is a mechanism
- 1792 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some

- extra data MAY be wrapped together with the DES key. This mechanism is defined in the SET protocol
- 1794 specifications.
- 1795 It takes a parameter, a **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure. This structure holds the
- 1796 "Block Contents" byte of the data and the concatenation of the hash of plaintext data (if present) and the
- extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by
- the *ulXLen* field having the value 0.
- When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present)
- and the extra data (if present) is returned following the convention described [PKCS #11-Curr],
- 1801 Miscellaneous simple key derivation mechanisms. Note that if the inputs to C\_UnwrapKey are such
- that the extra data is not returned (e.g. the buffer supplied in the
- 1803 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure is NULL\_PTR), then the unwrapped key object MUST
- 1804 NOT be created, either.
- Be aware that when this mechanism is used to unwrap a key, the *bBC* and *pX* fields of the parameter
- 1806 supplied to the mechanism MAY be modified.
- 1807 If an application uses **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP**, it may be preferable for it
- 1808 simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data
- 1809 (this concatenation MUST NOT be larger than 128 bytes), rather than calling **C\_UnwrapKey** twice. Each
- call of C\_UnwrapKey with CKM\_KEY\_WRAP\_SET\_OAEP requires an RSA decryption operation to be
- performed, and this computational overhead MAY be avoided by this means.
- 1812 **2.18 LYNKS**
- **2.18.1 Definitions**
- 1814 Mechanisms:

1824

1826

- 1815 CKM KEY WRAP LYNKS
- 1816 2.18.2 LYNKS key wrapping
- The LYNKS key wrapping mechanism, denoted **CKM\_KEY\_WRAP\_LYNKS**, is a mechanism for
- wrapping and unwrapping secret keys with DES keys. It MAY wrap any 8-byte secret key, and it produces
- 1819 a 10-byte wrapped key, containing a cryptographic checksum.
- 1820 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:
- 1822 1. Initialize two 16-bit integers, sum<sub>1</sub> and sum<sub>2</sub>, to 0
- 1823 2. Loop through the bytes of *K* from first to last.
  - 3. Set sum<sub>1</sub>= sum<sub>1</sub>+the key byte (treat the key byte as a number in the range 0-255).
- 1825 4. Set  $sum_2 = sum_2 + sum_1$ .
  - 5. Encrypt K with W in ECB mode, obtaining an encrypted key, E.
- 1827 6. Concatenate the last 6 bytes of E with sum<sub>2</sub>, representing sum<sub>2</sub> 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*.
- 1830 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.

1834	3 PKCS #11 Implementation Conformance
1835 1836	An implementation is a conforming implementation if it meets the conditions specified in one or more server profiles specified in <b>[PKCS #11-Prof]</b> .
1837	A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.
1838 1839 1840	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.

## Appendix A. Acknowledgments

- The following individuals have participated in the creation of this specification and are gratefully
- 1843 acknowledged:
- **Participants:**

1841

- 1845 Benton Stark Cisco Systems
- 1846 Anthony Berglas Cryptsoft Pty Ltd.
- 1847 Justin Corlett Cryptsoft Pty Ltd.
- 1848 Tony Cox Cryptsoft Pty Ltd.
- 1849 Tim Hudson Cryptsoft Pty Ltd.
- 1850 Bruce Rich Cryptsoft Pty Ltd.
- 1851 Greg Scott Cryptsoft Pty Ltd.
- 1852 Jason Thatcher Cryptsoft Pty Ltd.
- 1853 Magda Zdunkiewicz Cryptsoft Pty Ltd.
- 1854 Andrew Byrne Dell
- 1855 David Horton Dell
- 1856 Kevin Mooney Fornetix
- 1857 Gerald Stueve Fornetix
- 1858 Charles White Fornetix
- 1859 Matt Bauer Galois, Inc.
- 1860 Wan-Teh Chang Google Inc.
- 1861 Patrick Steuer IBM
- 1862 Michele Drgon Individual
- 1863 Gershon Janssen Individual
- 1864 Oscar So Individual
- 1865 Michelle Brochmann Information Security Corporation
- 1866 Michael Mrkowitz Information Security Corporation
- 1867 Jonathan Schulze-Hewett Information Security Corporation
- 1868 Philip Lafrance ISARA Corporation
- 1869 Thomas Hardjono M.I.T.
- 1870 Hamish Cameron nCipher
- 1871 Paul King nCipher
- 1872 Sander Temme nCipher
- 1873 Chet Ensign OASIS
- 1874 Jane Harnad OASIS
- 1875 Web Master OASIS
- 1876 Dee Schur OASIS
- 1877 Xuelei Fan Oracle
- 1878 Jan Friedel Oracle
- 1879 Susan Gleeson Oracle
- 1880 Dina Kurktchi-Nimeh Oracle
- 1881 John Leser Oracle

- 1882 Darren Moffat Oracle
- 1883 Mark Joseph P6R, Inc
- 1884 Jim Susoy P6R, Inc
- 1885 Roland Bramm PrimeKey Solutions AB
- 1886 Warren Armstrong QuintessenceLabs Pty Ltd.
- 1887 Kenli Chong QuintessenceLabs Pty Ltd.
- 1888 John Leiseboer QuintessenceLabs Pty Ltd.
- 1889 Florian Poppa QuintessenceLabs Pty Ltd.
- 1890 Martin Shannon QuintessenceLabs Pty Ltd.
- 1891 Jakub Jelen Red Hat
- 1892 Chris Malafis Red Hat
- 1893 Robert Relyea Red Hat
- 1894 Christian Bollich Utimaco IS GmbH
- 1895 Dieter Bong Utimaco IS GmbH
- 1896 Chris Meyer Utimaco IS GmbH
- 1897 Daniel Minder Utimaco IS GmbH
- 1898 Roland Reichenberg Utimaco IS GmbH
- 1899 Manish Upasani Utimaco IS GmbH
- 1900 Steven Wierenga Utimaco IS GmbH

# **Appendix B. Manifest constants**

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

- include/pkcs11-v3.0/pkcs11.h
- 1905 include/pkcs11-v3.0/pkcs11t.h

1901

1904

1906 • include/pkcs11-v3.0/pkcs11f.h

These files are linked from the "Additional artifacts" section at the top of this specification.

# **Appendix C. Revision History**

1909

1908

Revision	Date	Editor	Changes Made
wd01	20 April 2019	Dieter Bong	<ul> <li>All CAST5 definitions removed</li> <li>Replaced reference to [PKCS11-Base] table 10 by [PKCS11-Base] table 11 throughout whole document</li> </ul>
wd02	May 28, 2019	Tony Cox	Final cleanup of front introductory texts and links prior to CSPRD

1910