



PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40

Committee Specification Draft **0402** /
Public Review Draft **0402**

~~30 October 2013~~

23 April 2014

Specification URIs

This version:

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

Previous version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.doc> ~~N/A~~
(Authoritative)
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.html>
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd01/pkcs11-hist-v2.40-csprd01.pdf>

Latest version:

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

Technical Committee:

OASIS PKCS 11 TC

Chairs:

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

Editors:

Susan Gleeson (susan.gleeson@oracle.com), Oracle
Chris Zimman (czimman@bloomberg.com), Bloomberg Finance L.P.

Related work:

This specification is related to:

- *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*. Edited by [Susan Gleeson and Chris Zimman](#). Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40*. Edited by [Susan Gleeson and Chris Zimman](#). Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html>.

- *PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40*. Edited by [John Leiseboer and Robert Griffin](#). Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Profiles Version 2.40*. Edited by [Tim Hudson](#). Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html>.

Abstract:

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

Status:

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

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

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

Citation format:

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

[PKCS11-histHist-v2.40]

PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40. ~~30 October 2013~~. Edited by [Susan Gleeson and Chris Zimman](#). ~~23 April 2014~~. OASIS Committee Specification Draft ~~0402~~ / Public Review Draft ~~02~~. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csprd02/pkcs11-hist-v2.40-csprd02.html>~~04-~~. Latest version: <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>.

Notices

Copyright © OASIS Open 2013~~4~~. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

1	Introduction.....	8
1.1	Description of this Document.....	8
1.2	Terminology.....	8
1.3	Definitions.....	8
1.4	Normative References.....	9
1.5	Non-Normative References.....	9
2	Mechanisms.....	13
2.1	PKCS #11 Mechanisms.....	13
2.2	FORTEZZA timestamp.....	16
2.3	KEA.....	16
2.3.1	Definitions.....	16
2.3.2	KEA mechanism parameters.....	16
2.3.3	KEA public key objects.....	17
2.3.4	KEA private key objects.....	18
2.3.5	KEA key pair generation.....	18
2.3.6	KEA key derivation.....	19
2.4	RC2.....	20
2.4.1	Definitions.....	20
2.4.2	RC2 secret key objects.....	20
2.4.3	RC2 mechanism parameters.....	21
2.4.4	RC2 key generation.....	22
2.4.5	RC2-ECB.....	22
2.4.6	RC2-CBC.....	23
2.4.7	RC2-CBC with PKCS padding.....	23
2.4.8	General-length RC2-MAC.....	24
2.4.9	RC2-MAC.....	24
2.5	RC4.....	25
2.5.1	Definitions.....	25
2.5.2	RC4 secret key objects.....	25
2.5.3	RC4 key generation.....	25
2.5.4	RC4 mechanism.....	26
2.6	RC5.....	26
2.6.1	Definitions.....	26
2.6.2	RC5 secret key objects.....	26
2.6.3	RC5 mechanism parameters.....	27
2.6.4	RC5 key generation.....	28
2.6.5	RC5-ECB.....	28
2.6.6	RC5-CBC.....	29
2.6.7	RC5-CBC with PKCS padding.....	30
2.6.8	General-length RC5-MAC.....	30
2.6.9	RC5-MAC.....	31
2.7	General block cipher.....	31
2.7.1	Definitions.....	31

2.7.2 DES secret key objects	32
2.7.3 CAST secret key objects	33
2.7.4 CAST3 secret key objects	33
2.7.5 CAST128 (CAST5) secret key objects	34
2.7.6 IDEA secret key objects	34
2.7.7 CDMF secret key objects	35
2.7.8 General block cipher mechanism parameters.....	35
2.7.9 General block cipher key generation.....	36
2.7.10 General block cipher ECB.....	36
2.7.11 General block cipher CBC.....	37
2.7.12 General block cipher CBC with PKCS padding.....	37
2.7.13 General-length general block cipher MAC	38
2.7.14 General block cipher MAC	38
2.8 SKIPJACK.....	39
2.8.1 Definitions.....	39
2.8.2 SKIPJACK secret key objects	39
2.8.3 SKIPJACK Mechanism parameters	40
2.8.4 SKIPJACK key generation	42
2.8.5 SKIPJACK-ECB64	42
2.8.6 SKIPJACK-CBC64	42
2.8.7 SKIPJACK-OFB64	43
2.8.8 SKIPJACK-CFB64.....	43
2.8.9 SKIPJACK-CFB32.....	43
2.8.10 SKIPJACK-CFB16.....	43
2.8.11 SKIPJACK-CFB8.....	44
2.8.12 SKIPJACK-WRAP	44
2.8.13 SKIPJACK-PRIVATE-WRAP	44
2.8.14 SKIPJACK-RELAYX.....	44
2.9 BATON.....	44
2.9.1 Definitions.....	44
2.9.2 BATON secret key objects	45
2.9.3 BATON key generation	46
2.9.4 BATON-ECB128	46
2.9.5 BATON-ECB96.....	46
2.9.6 BATON-CBC128	46
2.9.7 BATON-COUNTER	47
2.9.8 BATON-SHUFFLE	47
2.9.9 BATON WRAP	47
2.10 JUNIPER.....	47
2.10.1 Definitions.....	47
2.10.2 JUNIPER secret key objects	48
2.10.3 JUNIPER key generation	48
2.10.4 JUNIPER-ECB128	49
2.10.5 JUNIPER-CBC128	49
2.10.6 JUNIPER-COUNTER	49

2.10.7 JUNIPER-SHUFFLE	50
2.10.8 JUNIPER WRAP	50
2.11 MD2	50
2.11.1 Definitions	50
2.11.2 MD2 digest	50
2.11.3 General-length MD2-HMAC	50
2.11.4 MD2-HMAC	51
2.11.5 MD2 key derivation	51
2.12 MD5	52
2.12.1 Definitions	52
2.12.2 MD5 Digest	52
2.12.3 General-length MD5-HMAC	52
2.12.4 MD5-HMAC	52
2.12.5 MD5 key derivation	52
2.13 FASTHASH	53
2.13.1 Definitions	53
2.13.2 FASTHASH digest	53
2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)	53
2.14.1 Definitions	53
2.14.2 Password-based encryption/authentication mechanism parameters	54
2.14.3 MD2-PBE for DES-CBC	55
2.14.4 MD5-PBE for DES-CBC	55
2.14.5 MD5-PBE for CAST-CBC	55
2.14.6 MD5-PBE for CAST3-CBC	55
2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC)	55
2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)	56
2.15 PKCS #12 password-based encryption/authentication mechanisms	56
2.15.1 Definitions	56
2.15.2 SHA-1-PBE for 128-bit RC4	57
2.15.3 SHA-1_PBE for 40-bit RC4	57
2.15.4 SHA-1_PBE for 128-bit RC2-CBC	57
2.15.5 SHA-1_PBE for 40-bit RC2-CBC	57
2.16 RIPE-MD	58
2.16.1 Definitions	58
2.16.2 RIPE-MD 128 Digest	58
2.16.3 General-length RIPE-MD 128-HMAC	58
2.16.4 RIPE-MD 128-HMAC	58
2.16.5 RIPE-MD 160	59
2.16.6 General-length RIPE-MD 160-HMAC	59
2.16.7 RIPE-MD 160-HMAC	59
2.17 SET	59
2.17.1 Definitions	59
2.17.2 SET mechanism parameters	59
2.17.3 OAEP key wrapping for SET	60
2.18 LYNKS	60

2.18.1	Definitions	60
2.18.2	LYNKS key wrapping	60
3	PKCS #11 Implementation Conformance	62
Appendix A.	Acknowledgments	63
Appendix B.	Manifest constants	65
Appendix C.	Revision History	68

1 Introduction

1.1 Description of this Document

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

All text is normative unless otherwise labeled.

1.1.2 Terminology

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

1.2.1.3 Definitions

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

BATON	MISSI's BATON block cipher.
CAST	Entrust Technologies' proprietary symmetric block cipher
CAST3	Entrust Technologies' proprietary symmetric block cipher
CAST5	Another name for Entrust Technologies' symmetric block cipher CAST128. CAST128 is the preferred name.
CAST128	Entrust Technologies' symmetric block cipher.
CDMF	Commercial Data Masking Facility, a block encipherment method specified by International Business Machines Corporation and based on DES.
CMS	Cryptographic Message Syntax (see RFC 26303369)
DES	Data Encryption Standard, as defined in FIPS PUB 46-3
ECB	Electronic Codebook mode, as defined in FIPS PUB 81.
FASTHASH	MISSI's FASTHASH message-digesting algorithm.
IDEA	Ascom Systec's symmetric block cipher.
IV	Initialization Vector.
JUNIPER	MISSI's JUNIPER block cipher.
KEA	MISSI's Key Exchange Algorithm.
LYNKS	A smart card manufactured by SPYRUS.
MAC	Message Authentication Code
MD2	RSA Security's MD2 message-digest algorithm, as defined in RFC 613449 .

37	MD5	RSA Security's MD5 message-digest algorithm, as defined in RFC
38		1321.
39	PRF	Pseudo random function.
40	RSA	The RSA public-key cryptosystem.
41	RC2	RSA Security's RC2 symmetric block cipher.
42	RC4	RSA Security's proprietary RC4 symmetric stream cipher.
43	RC5	RSA Security's RC5 symmetric block cipher.
44	SET	The Secure Electronic Transaction protocol.
45	SHA-1	The (revised) Secure Hash Algorithm with a 160-bit message digest,
46		as defined in FIPS PUB 180-2.
47	SKIPJACK	MISSI's SKIPJACK block cipher.

49 ~~**UTF-8** Universal Character Set (UCS) transformation format (UTF) that~~
50 ~~represents ISO 10646 and UNICODE strings with a variable number~~
51 ~~of octets~~

52 **1.31.4 Normative References**

53	[PKCS #11-Base]	<i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i> Latest
54		version. http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-
55		v2.40.html .
56		
57	[PKCS #11-Curr]	<i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification</i>
58		<i>Version 2.40.</i> Latest version. http://docs.oasis-open.org/pkcs11/pkcs11-
59		curr/v2.40/pkcs11-curr-v2.40.html .
60		
61	[PKCS #11-Prof]	<i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i> Latest version.
62		http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-
63		v2.40.html .
64		
65	[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP
66		14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt .
67		

68 **1.41.5 Non-Normative References**

69	[ANSI C]	ANSI/ISO. <i>American National Standard for Programming Languages – C.</i> 1990
70	[ANSI X9.31]	Accredited Standards Committee X9. <i>Digital Signatures Using Reversible Public</i>
71		<i>Key Cryptography for the Financial Services Industry (rDSA).</i> 1998.
72	[ANSI X9.42]	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial</i>
73		<i>Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm</i>
74		<i>Cryptography.</i> 2003
75	[ANSI X9.62]	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial</i>
76		<i>Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).</i> 1998
77	[CC/PP]	W3C-G. Klyne, F. Reynolds, C., H. Ohto, J. Hjelm, M. H. Butler, L. Tran, Editors,
78		W3C. Composite Capability/Preference Profiles (CC/PP): Structure and
79		Vocabularies. World Wide Web Consortium, January 2004. URL:
80		http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/
81	[CDPD]	Ameritech Mobile Communications et al. <i>Cellular Digital Packet Data System</i>
82		<i>Specifications: Part 406: Airlink Security.</i> 1993

- 83 [FIPS PUB 46-3] NIST. *FIPS 46-3: Data Encryption Standard (DES)*. October 26, 1999. URL:
84 <http://csrc.nist.gov/publications/fips/index.html>
- 85 ~~[FIPS PUB 74] NIST. *FIPS 74: Guidelines for Implementing and Using the NBS Data Encryption*~~
86 ~~*Standard*. April 1, 1981. URL:~~ [FIPS PUB 81] NIST. *FIPS 81: DES Modes of*
87 *Operation*. December 1980. URL:
88 <http://csrc.nist.gov/publications/fips/index.html>
- 89 [FIPS PUB 113] NIST. *FIPS 113: Computer Data Authentication*. May 30, 1985. URL:
90 <http://csrc.nist.gov/publications/fips/index.html>
- 91 [FIPS PUB 180-2] NIST. *FIPS 180-2: Secure Hash Standard*. August 1, 2002. URL:
92 <http://csrc.nist.gov/publications/fips/index.html>
- 93 ~~[FIPS PUB 186-2] NIST. *FIPS 186-2: Digital Signature Standard*. January 27, 2000. URL:~~
94 ~~[FIPS PUB 197] NIST. *FIPS 197: Advanced Encryption Standard (AES)*. November 26, 2001.~~
95 ~~URL:~~ [FORTEZZA CIPG] NSA, Workstation Security Products.
96 *FORTEZZA Cryptologic Interface Programmers Guide, Revision 1.52*.
97 November 1985
- 98 [GCS-API] X/Open Company Ltd. *Generic Cryptographic Service API (GCS-API), Base –*
99 *Draft 2*. February 14, 1995.
- 100 [ISO/IEC 7816-1] ISO. ~~*Information Technology – IEC 7816-1:2011. Identification Cards –*~~
101 ~~*Integrated Circuit(s) with Contacts – circuit cards -- Part 1: Cards with contacts --*~~
102 ~~*Physical Characteristics*. 1998-2011 URL:~~
103 http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089.
- 104 [ISO/IEC 7816-4] ISO. ~~*Information Technology – IEC 7618-4:2013. Identification Cards –*~~
105 ~~*Integrated Circuit(s) with Contacts circuit cards – Part 4: Interindustry*~~
106 ~~*Commands Organization, security and commands for interchange*. 2013. URL:~~
107 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnum](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54550)
108 ~~[ber=54550](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54550)~~ ~~*Interchange*. 1995.~~
- 109 [ISO/IEC 8824-1] ISO. ~~*Information Technology – IEC 8824-1:2008. Abstract Syntax Notation One*~~
110 ~~*(ASN.1): Specification of Base Notation*. 2002. URL:~~
111 [http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012)
112 [54012](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012)
- 113 [ISO/IEC 8825-1] ISO. ~~*IEC 8825-1:2008. Information Technology – ASN.1 Encoding Rules:*~~
114 ~~*Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER),*~~
115 ~~*and Distinguished Encoding Rules (DER)*. 2002-2008. URL:~~
116 [http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnum](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60)
117 ~~[ber=54011&ics1=35&ics2=100&ics3=60](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60)~~
- 118 [ISO/IEC 9594-1] ISO. ~~*IEC 9594-1:2008. Information Technology – Open System Interconnection –*~~
119 ~~*The Directory: Overview of Concepts, Models and Services*. 2004-2008. URL:~~
120 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnum](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53364)
121 ~~[ber=53364](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53364)~~
- 122 [ISO/IEC 9594-8] ISO. ~~*IEC 9594-8:2008. Information Technology – Open Systems Interconnection*~~
123 ~~*– The Directory: Public-key and Attribute Certificate Frameworks*. 2004-2008~~
124 ~~URL:~~
125 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnum](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53372)
126 ~~[ber=53372](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53372)~~
- 127 [ISO/IEC 9796-2] ISO. ~~*IEC 9796-2:2010. Information Technology – Security Techniques – Digital*~~
128 ~~*Signature Scheme Giving Message Recovery – Part 2: Integer factorization*~~
129 ~~*based mechanisms*. 2002-2010. URL:~~
130 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnum](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788)
131 ~~[ber=54788](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788)~~
- 132 [Java MIDP] Java Community Process. *Mobile Information Device Profile for Java 2 Micro*
133 *Edition*. November 2002. URL: <http://jcp.org/jsr/detail/118.jsp>
- 134 [MeT-PTD] MeT. *MeT PTD Definition – Personal Trusted Device Definition, Version 1.0*.
135 February 2003. URL: <http://www.mobiletransaction.org>

- 136 **[PCMCIA]** Personal Computer Memory Card International Association. *PC Card Standard*,
137 *Release 2.1*. July 1993.
- 138 **[PKCS #1]** RSA Laboratories. *RSA Cryptography Standard, v2.1*. June 14, 2002 URL:
139 <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>
- 140 **[PKCS #3]** RSA Laboratories. *Diffie-Hellman Key-Agreement Standard, v1.4*. November
141 1993.
- 142 **[PKCS #5]** RSA Laboratories. *Password-Based Encryption Standard, v2.0*. March 26,
143 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf>
- 144 **[PKCS #7]** RSA Laboratories. *Cryptographic Message Syntax Standard, v1.56*. November
145 ~~1993~~1997 URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v16.pdf>
- 146 **[PKCS #8]** RSA Laboratories. *Private-Key Information Syntax Standard, v1.2*. November
147 1993. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn
- 148 **[PKCS #11-UG]** *PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40*. Latest
149 version. [http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-](http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html)
150 [v2.40.html](http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html).
- 151 ~~**[PKCS #11-C]** RSA Laboratories. *PKCS#11: Conformance Profile Specification*. October
152 2000.~~
- 153 ~~**[PKCS #11-P]** RSA Laboratories. *PKCS #11 Profiles for mobile devices*. June 2003.~~
- 154
- 155 **[PKCS #12]** RSA Laboratories. *Personal Information Exchange Syntax Standard, v1.0*.
156 June 1999.
- 157 ~~**[RFC 1319]** B. Kaliski. *RFC 1319: The MD2 Message-Digest Algorithm*. RSA Laboratories,
158 April 1992. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf>~~
- 159 **[RFC 1321]** R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. MIT Laboratory for
160 Computer Science and RSA Data Security, Inc., April 1992. URL:
161 <http://www.rfc-editor.org/rfc/rfc1321.txt>
- 162 ~~**[RFC 1421]** J. Linn. *RFC 1421: Privacy Enhancement for Internet Electronic Mail: Part I:*
163 *Message Encryption and Authentication Procedures*. IAB IRTF PSRG, IETF
164 PEM WG, February 1993. URL:~~
- 165 ~~**[RFC 2045]** Freed, N., and Borenstein. *RFC 2045: Multipurpose Internet Mail Extensions*
166 *(MIME) Part One: Format of Internet Message Bodies*. November 1996. URL:~~
- 167 ~~**[RFC 2246]** T. Dierks and C. Allen. *RFC 2245: The TLS Protocol Version 1.0*. Certicom,
168 January 1999. URL:~~
- 169 ~~**[RFC 2279]** F. Yergeau. *RFC 2279: UTF-8, a transformation format of ISO 10646*. Alis
170 Technologies, January 1998. URL:~~
- 171 ~~**[RFC 2534]** Masinter, L., Wing, D., Mutz, A., and K. Holtman. *RFC 2534: Media Features for*
172 *Display, Print and Fax*. March 1999. URL:~~
- 173 ~~**[RFC 2630 3369]** R. Houseley. *RFC 2630: cryptographic3369: Cryptographic Message Syntax-*
174 *June 1999. (CMS). August 2002*. URL: [http://ietfwww.rfc-](http://ietfwww.rfc-editor.org/rfc/rfc2630369.txt)
175 [editor.org/rfc/rfc2630369.txt](http://ietfwww.rfc-editor.org/rfc/rfc2630369.txt)~~
- 176 ~~**[RFC 2743]** J. Linn. *RFC 2743: Generic Security Service Application Program Interface*
177 *Version 2, Update 1*. RSA Laboratories, January 2000. URL:~~
- 178 ~~**[RFC 2744]** J. Wray. *RFC 2744: Generic Security Services API Version 2: C-bindings*. Iris
179 Associates, January 2000. URL: ~~[\[RFC 6149\]](#)~~ S. Turner and L. Chen. *RFC*
180 *6149: MD2 to Historic Status*. March, 2011. URL: [http://www.rfc-](http://www.rfc-editor.org/rfc/rfc6149.txt)
181 [editor.org/rfc/rfc6149.txt](http://www.rfc-editor.org/rfc/rfc6149.txt)~~
- 182 **[SEC-1]** Standards for Efficient Cryptography Group (SECG). *Standards for Efficient*
183 *Cryptography (SEC) 1: Elliptic Curve Cryptography*. Version 1.0, September 20,
184 2000.
- 185 **[SEC-2]** Standards for Efficient cryptography Group (SECG). *Standards for Efficient*
186 *Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters*.
187 Version 1.0, September 20, 2000.

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

190 [WIM] WAP. *Wireless Identity Module*. – WAP-260-WIP-20010712.a. July 2001. URL:
191 <http://www.wapforumtechnical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf>
192

193 [WPKI] WAP. *Wireless PKI Application Protocol: Public Key Infrastructure Definition*. –
194 WAP-217-WPKI-20010424-a. April 2001. URL:
195 <http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf>
196

197 [WTLS] WAP. *Wireless Transport Layer Security Version* – WAP-261-WTLS-20010406-
198 a. April 2001. URL:
199 <http://www.wapforumtechnical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf>
200

201 [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)
202
203

204 [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)
205
206

207 [X.680] ITU-T. *Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. July 2002. (Identical to ISO/IEC 8824-1)
208

209 [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)
210
211
212

213

2 Mechanisms

214

2.1 PKCS #11 Mechanisms

215

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.

217

218

219

220

221

222

223

224

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

225

Table 1, Mechanisms vs. Functions

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

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_CAST_MAC_GENERAL		X					
CKM_CAST_MAC		X					
CKM_CAST3_KEY_GEN					X		
CKM_CAST3_ECB	X					X	
CKM_CAST3_CBC	X					X	
CKM_CAST3_CBC_PAD	X					X	
CKM_CAST3_MAC_GENERAL		X					
CKM_CAST3_MAC		X					
CKM_CAST128_KEY_GEN (CKM_CAST5_KEY_GEN)					X		
CKM_CAST128_ECB (CKM_CAST5_ECB)	X					X	
CKM_CAST128_CBC (CKM_CAST5_CBC)	X					X	
CKM_CAST128_CBC_PAD (CKM_CAST5_CBC_PAD)	X					X	
CKM_CAST128_MAC_GENERAL (CKM_CAST5_MAC_GENERAL)		X					
CKM_CAST128_MAC (CKM_CAST5_MAC)		X					
CKM_IDEA_KEY_GEN					X		
CKM_IDEA_ECB	X					X	
CKM_IDEA_CBC	X					X	
CKM_IDEA_CBC_PAD	X					X	
CKM_IDEA_MAC_GENERAL		X					
CKM_IDEA_MAC		X					
CKM_CDMF_KEY_GEN					X		
CKM_CDMF_ECB	X					X	
CKM_CDMF_CBC	X					X	
CKM_CDMF_CBC_PAD	X					X	
CKM_CDMF_MAC_GENERAL		X					
CKM_CDMF_MAC		X					
CKM_SKIPJACK_KEY_GEN					X		
CKM_SKIPJACK_ECB64	X						
CKM_SKIPJACK_CBC64	X						
CKM_SKIPJACK_OFB64	X						
CKM_SKIPJACK_CFB64	X						
CKM_SKIPJACK_CFB32	X						
CKM_SKIPJACK_CFB16	X						
CKM_SKIPJACK_CFB8	X						
CKM_SKIPJACK_WRAP						X	
CKM_SKIPJACK_PRIVATE_WRAP						X	
CKM_SKIPJACK_RELAYX						X ³	
CKM_BATON_KEY_GEN					X		
CKM_BATON_ECB128	X						
CKM_BATON_ECB96	X						

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_BATON_CBC128	X						
CKM_BATON_COUNTER	X						
CKM_BATON_SHUFFLE	X						
CKM_BATON_WRAP						X	
CKM_JUNIPER_KEY_GEN					X		
CKM_JUNIPER_ECB128	X						
CKM_JUNIPER_CBC128	X						
CKM_JUNIPER_COUNTER	X						
CKM_JUNIPER_SHUFFLE	X						
CKM_JUNIPER_WRAP						X	
CKM_MD2				X			
CKM_MD2_HMAC_GENERAL		X					
CKM_MD2_HMAC		X					
CKM_MD2_KEY_DERIVATION							X
CKM_MD5				X			
CKM_MD5_HMAC_GENERAL		X					
CKM_MD5_HMAC		X					
CKM_MD5_KEY_DERIVATION							X
CKM_RIPEMD128				X			
CKM_RIPEMD128_HMAC_GENERAL		X					
CKM_RIPEMD128_HMAC		X					
CKM_RIPEMD160				X			
CKM_RIPEMD160_HMAC_GENERAL		X					
CKM_RIPEMD160_HMAC		X					
CKM_FASTHASH				X			
CKM_PBE_MD2_DES_CBC					X		
CKM_PBE_MD5_DES_CBC					X		
CKM_PBE_MD5_CAST_CBC					X		
CKM_PBE_MD5_CAST3_CBC					X		
CKM_PBE_MD5_CAST128_CBC (CKM_PBE_MD5_CAST5_CBC)					X		
CKM_PBE_SHA1_CAST128_CBC (CKM_PBE_SHA1_CAST5_CBC)					X		
CKM_PBE_SHA1_RC4_128					X		
CKM_PBE_SHA1_RC4_40					X		
CKM_PBE_SHA1_RC2_128_CBC					X		
CKM_PBE_SHA1_RC2_40_CBC					X		
CKM_PBA_SHA1_WITH_SHA1_HMAC					X		
CKM_PKCS5_PBKD2					X		
CKM_KEY_WRAP_SET_OAEP						X	
CKM_KEY_WRAP_LYNKS						X	

226 ¹ SR = SignRecover, VR = VerifyRecover.

227 ² Single-part operations only.

228 ³ Mechanism ~~can~~**MUST** only be used for wrapping, not unwrapping.

229 | The remainder of this section ~~will present~~**presents** in detail the mechanisms supported by Cryptoki and
230 | the parameters which are supplied to them.

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

233

234 | **2.12.2 FORTEZZA timestamp**

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

238 | **It has no parameters.**

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

241 | *Table 2, FORTEZZA Timestamp: Key and Data Length*

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

242 | ¹ Single-part operations only

243 | ² Data length, signature length

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

246 | **2.22.3 KEA**

247 | **2.2.12.3.1 Definitions**

248 | This section defines the key type "CKK_KEA" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
249 | attribute of key objects.

250 | Mechanisms:

251 | CKM_KEA_KEY_PAIR_GEN

252 | CKM_KEA_KEY_DERIVE

253 | **2.2.22.3.2 KEA mechanism parameters**

254 | **2.2.2.12.3.2.1 CK_KEA_DERIVE_PARAMS; CK_KEA_DERIVE_PARAMS_PTR**

255

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

```
258 | typedef struct CK_KEA_DERIVE_PARAMS {  
259 |     CK_BBOOL isSender;  
260 |     CK_ULONG ulRandomLen;  
261 |     CK_BYTE_PTR pRandomA;  
262 |     CK_BYTE_PTR pRandomB;  
263 |     CK_ULONG ulPublicDataLen;  
264 |     CK_BYTE_PTR pPublicData;  
265 | } CK_KEA_DERIVE_PARAMS;
```

266

267 | The fields of the structure have the following meanings:

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

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

272 *pRandomA* pointer to Ra data

273 *pRandomB* pointer to Rb data

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

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

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

277 **2.2.32.3.3 KEA public key objects**

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

281 *Table 3, KEA Public Key Object Attributes*

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

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

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

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

```

286 CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
287 CK_KEY_TYPE keyType = CKK_KEA;
288 CK_UTF8CHAR label[] = "A KEA public key object";
289 CK_BYTE prime[] = {...};
290 CK_BYTE subprime[] = {...};
291 CK_BYTE base[] = {...};
292 CK_BYTE value[] = {...};
293 CK_ATTRIBUTE template[] = {
294     {CKA_CLASS, &class, sizeof(class)},
295     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
296     {CKA_TOKEN, &true, sizeof(true)},
297     {CKA_LABEL, label, sizeof(label)-1},
298     {CKA_PRIME, prime, sizeof(prime)},
299     {CKA_SUBPRIME, subprime, sizeof(subprime)},
300     {CKA_BASE, base, sizeof(base)},
301     {CKA_VALUE, value, sizeof(value)}
302 };
  
```

303

304 2.2.42.3.4 KEA private key objects

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

308 Table 4, KEA Private Key Object Attributes

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

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

310

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

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

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

```
317 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;  
318 CK_KEY_TYPE keyType = CKK_KEA;  
319 CK_UTF8CHAR label[] = "A KEA private key object";  
320 CK_BYTE subject[] = {...};  
321 CK_BYTE id[] = {123};  
322 CK_BYTE prime[] = {...};  
323 CK_BYTE subprime[] = {...};  
324 CK_BYTE base[] = {...};  
325 CK_BYTE value[] = {...};  
326 CK_BBOOL true = CK_TRUE;  
327 CK_ATTRIBUTE template[] = {  
328     {CKA_CLASS, &class, sizeof(class)},  
329     {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS  
330     {CKA_TOKEN, &>true, sizeof(true)},  
331     {CKA_LABEL, label, sizeof(label) - 1},  
332     {CKA_SUBJECT, subject, sizeof(subject)},  
333     {CKA_ID, id, sizeof(id)},  
334     {CKA_SENSITIVE, &>true, sizeof(true)},  
335     {CKA_DERIVE, &>true, sizeof(true)},  
336     {CKA_PRIME, prime, sizeof(prime)},  
337     {CKA_SUBPRIME, subprime, sizeof(subprime)},  
338     {CKA_BASE, base, sizeof(base)},  
339     {CKA_VALUE, value, sizeof(value)}  
340 };
```

341 2.2.52.3.5 KEA key pair generation

342 The KEA key pair generation mechanism, denoted **CKM_KEA_KEY_PAIR_GEN**, generates key pairs for
343 the Key Exchange Algorithm, as defined by NIST’s “SKIPJACK and KEA Algorithm Specification Version
344 2.0”, 29 May 1998.

345 It does not have a parameter.

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

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

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

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

357 **2.2.62.3.6 KEA key derivation**

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

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

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

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

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

377 *Table 5, KEA Parameter Values and Operations*

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

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

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

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

- 383 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key
384 ~~can~~**MAY** both be specified to be either CK_TRUE or CK_FALSE. If omitted, these attributes each
385 take on some default value.
- 386 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, then the derived
387 key ~~will~~**MUST** as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to
388 CK_TRUE, then the derived has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value
389 as its **CKA_SENSITIVE** attribute.
- 390 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE, then
391 the derived key ~~will~~**MUST**, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute
392 set to CK_TRUE, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the
393 *opposite* value from its **CKA_EXTRACTABLE** attribute.

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

396 2.32.4 RC2

397 2.4.1 Definitions

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

403 **2.3.1 Definitions**

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

406 Mechanisms:

- 407 CKM_RC2_KEY_GEN
- 408 CKM_RC2_ECB
- 409 CKM_RC2_CBC
- 410 CKM_RC2_MAC
- 411 CKM_RC2_MAC_GENERAL
- 412 CKM_RC2_CBC_PAD

413 2.3.22.4.2 RC2 secret key objects

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

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

417 Table 6, RC2 Secret Key Object Attributes

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

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

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

```

420 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
421 CK_KEY_TYPE keyType = CKK_RC2;
422 CK_UTF8CHAR label[] = "An RC2 secret key object";
423 CK_BYTE value[] = {...};
424 CK_BBOOL true = CK_TRUE;
425 CK_ATTRIBUTE template[] = {
426     {CKA_CLASS, &class, sizeof(class)},
427     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
428     {CKA_TOKEN, &>true, sizeof(true)},
429     {CKA_LABEL, label, sizeof(label)-1},
430     {CKA_ENCRYPT, &>true, sizeof(true)},
431     {CKA_VALUE, value, sizeof(value)}
432 };

```

433 2.3.32.4.3 RC2 mechanism parameters

434 2.3.3-12.4.3.1 CK_RC2_PARAMS; CK_RC2_PARAMS_PTR

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

```

437 typedef CK_ULONG CK_RC2_PARAMS;

```

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

439 2.3.3-22.4.3.2 CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR

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

```

442 typedef struct CK_RC2_CBC_PARAMS {
443     CK_ULONG ulEffectiveBits;
444     CK_BYTE iv[8];
445 } CK_RC2_CBC_PARAMS;

```

446 The fields of the structure have the following meanings:

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

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

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

451 2.3.3-32.4.3.3 CK_RC2_MAC_GENERAL_PARAMS; 452 CK_RC2_MAC_GENERAL_PARAMS_PTR

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

```

455 typedef struct CK_RC2_MAC_GENERAL_PARAMS {

```

```

456     CK_ULONG ulEffectiveBits;
457     CK_ULONG ulMacLength;
458 } CK_RC2_MAC_GENERAL_PARAMS;

```

459 The fields of the structure have the following meanings:

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

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

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

463 **2.3.42.4.4 RC2 key generation**

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

466 It does not have a parameter.

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

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

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

474 **2.3.52.4.5 RC2-ECB**

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

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

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

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

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

491 *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	
-------------	-----	---------------	--	--

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

494 **2.3.62.4.6 RC2-CBC**

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

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

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

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

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

512 *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	

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

515 **2.3.72.4.7 RC2-CBC with PKCS padding**

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

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

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

525 | In addition to being able to wrap and unwrap secret keys, this mechanism ~~can~~**MAY** wrap and unwrap
 526 | RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see
 527 | ~~***MISSING REFERENCE***~~**[PKCS #11-Curr], Miscellaneous simple key derivation mechanisms** for
 528 | details). The entries in the table below for data length constraints when wrapping and unwrapping keys
 529 | do not apply to wrapping and unwrapping private keys.

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

531 | *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

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

534 | **2.3.82.4.8 General-length RC2-MAC**

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

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

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

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

543 | *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

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

546 | **2.3.92.4.9 RC2-MAC**

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

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

552

553 | *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
----------	-----	-----	---

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

556 **2.4.2.5 RC4**

557 **2.4.12.5.1 Definitions**

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

560 Mechanisms

561 CKM_RC4_KEY_GEN

562 CKM_RC4

563 **2.4.22.5.2 RC4 secret key objects**

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

567 *Table 12, RC4 Secret Key Object*

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

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

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

```
570 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
571 CK_KEY_TYPE keyType = CKK_RC4;
572 CK_UTF8CHAR label[] = "An RC4 secret key object";
573 CK_BYTE value[] = {...};
574 CK_BBOOL true = CK_TRUE;
575 CK_ATTRIBUTE template[] = {
576     {CKA_CLASS, &class, sizeof(class)},
577     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
578     {CKA_TOKEN, &>true, sizeof(true)},
579     {CKA_LABEL, label, sizeof(label)-1},
580     {CKA_ENCRYPT, &>true, sizeof(true)},
581     {CKA_VALUE, value, sizeof(value)}
582 };
```

583 **2.4.32.5.3 RC4 key generation**

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

586 It does not have a parameter.

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

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

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

595 **2.4.42.5.4 RC4 mechanism**

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

598 It does not have a parameter.

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

600 *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

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

603 **2.52.6 RC5**

604 **2.6.1 Definitions**

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

607 **2.5.11.1.1 Definitions**

608 This section defines the key type “CKK_RC5” for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
 609 attribute of key objects.

610 Mechanisms:

- 611 CKM_RC5_KEY_GEN
- 612 CKM_RC5_ECB
- 613 CKM_RC5_CBC
- 614 CKM_RC5_MAC
- 615 CKM_RC5_MAC_GENERAL
- 616 CMK_RC5_CBC_PAD

617 **2.5.22.6.2 RC5 secret key objects**

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

621 *Table 14, RC5 Secret Key Object*

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

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

623

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

```
625 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
626 CK_KEY_TYPE keyType = CKK_RC5;
627 CK_UTF8CHAR label[] = "An RC5 secret key object";
628 CK_BYTE value[] = {...};
629 CK_BBOOL true = CK_TRUE;
630 CK_ATTRIBUTE template[] = {
631     {CKA_CLASS, &class, sizeof(class)},
632     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
633     {CKA_TOKEN, &>true, sizeof(true)},
634     {CKA_LABEL, label, sizeof(label)-1},
635     {CKA_ENCRYPT, &>true, sizeof(true)},
636     {CKA_VALUE, value, sizeof(value)}
637 };
```

638 2.5.32.6.3 RC5 mechanism parameters

639 2.5.3.12.6.3.1 CK_RC5_PARAMS; CK_RC5_PARAMS_PTR

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

```
642 typedef struct CK_RC5_PARAMS {
643     CK_ULONG ulWordsize;
644     CK_ULONG ulRounds;
645 } CK_RC5_PARAMS;
```

646 The fields of the structure have the following meanings:

647 *ulWordsize* wordsize of RC5 cipher in bytes

648 *ulRounds* number of rounds of RC5 encipherment

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

650 2.5.3.22.6.3.2 CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR

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

```
653 typedef struct CK_RC5_CBC_PARAMS {
654     CK_ULONG ulWordsize;
655     CK_ULONG ulRounds;
656     CK_BYTE_PTR pIv;
657     CK_ULONG ulIvLen;
658 } CK_RC5_CBC_PARAMS;
```

659 The fields of the structure have the following meanings:

660 *ulwordSize* wordsize of RC5 cipher in bytes

661 *ulRounds* number of rounds of RC5 encipherment

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

663 *ulIvLen* length of initialization vector (must be same as
664 blocksize)

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

666 **2.5.3.32.6.3.3 CK_RC5_MAC_GENERAL_PARAMS;**
667 **CK_RC5_MAC_GENERAL_PARAMS_PTR**

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

```
670 typedef struct CK_RC5_MAC_GENERAL_PARAMS {  
671     CK_ULONG ulWordsize;  
672     CK_ULONG ulRounds;  
673     CK_ULONG ulMacLength;  
674 } CK_RC5_MAC_GENERAL_PARAMS;
```

675 The fields of the structure have the following meanings:

676 *ulwordSize* wordsize of RC5 cipher in bytes

677 *ulRounds* number of rounds of RC5 encipherment

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

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

680 **2.5.42.6.4 RC5 key generation**

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

683 It does not have a parameter.

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

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

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

691 **2.5.52.6.5 RC5-ECB**

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

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

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

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

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

708 *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	

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

711 2.5.62.6.6 RC5-CBC

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

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

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

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

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

728 *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	

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

731 **2.5.72.6.7 RC5-CBC with PKCS padding**

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

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

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

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

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

746 *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

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

749 **2.5.82.6.8 General-length RC5-MAC**

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

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

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

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

758 *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

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

761 **2.5.92.6.9 RC5-MAC**

762 RC5-MAC, denoted by **CKM_RC5_MAC**, is a special case of the general-length RC5-MAC mechanism.
763 Instead of taking a **CK_RC5_MAC_GENERAL_PARAMS** parameter, it takes a **CK_RC5_PARAMS**
764 parameter. RC5-MAC ~~always~~ produces and verifies MACs half as large as the RC5 blocksize.
765 Constraints on key types and the length of data are summarized in the following table:

766 *Table 19, RC5-MAC: Key and Data Length*

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

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

769 **2.6.2.7 General block cipher**

770 **2.7.1 Definitions**

771 For brevity's sake, the mechanisms for the DES, CAST, CAST3, CAST128 (CAST5), IDEA and CDMF
772 block ciphers ~~will beare~~ described together here. Each of these ciphers ha the following mechanisms,
773 which ~~will beare~~ described in a templated form.

774 **2.6.11.1.1 Definitions**

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

778 Mechanisms:

- 779 CKM_DES_KEY_GEN
- 780 CKM_DES_ECB
- 781 CKM_DES_CBC
- 782 CKM_DES_MAC
- 783 CKM_DES_MAC_GENERAL
- 784 CKM_DES_CBC_PAD
- 785 CKM_CDMF_KEY_GEN
- 786 CKM_CDMF_ECB
- 787 CKM_CDMF_CBC
- 788 CKM_CDMF_MAC
- 789 CKM_CDMF_MAC_GENERAL
- 790 CKM_CDMF_CBC_PAD
- 791 CKM_DES_OFB64
- 792 CKM_DES_OFB8
- 793 CKM_DES_CFB64
- 794 CKM_DES_CFB8
- 795 CKM_CAST_KEY_GEN
- 796 CKM_CAST_ECB
- 797 CKM_CAST_CBC

798 CKM_CAST_MAC
 799 CKM_CAST_MAC_GENERAL
 800 CKM_CAST_CBC_PAD
 801 CKM_CAST3_KEY_GEN
 802 CKM_CAST3_ECB
 803 CKM_CAST3_CBC
 804 CKM_CAST3_MAC
 805 CKM_CAST3_MAC_GENERAL
 806 CKM_CAST3_CBC_PAD
 807 CKM_CAST5_KEY_GEN
 808 CKM_CAST128_KEY_GEN
 809 CKM_CAST5_ECB
 810 CKM_CAST128_ECB
 811 CKM_CAST5_CBC
 812 CKM_CAST128_CB C
 813 CKM_CAST5_MAC
 814 CKM_CAST128_MAC
 815 CKM_CAST5_MAC_GENERAL
 816 CKM_CAST128_MAC_GENERAL
 817 CKM_CAST5_CBC_PAD
 818 CKM_CAST128_CBC_PAD
 819 CKM_IDEA_KEY_GEN
 820 CKM_IDEA_ECB
 821 CKM_IDEA_MAC
 822 CKM_IDEA_MAC_GENERAL
 823 CKM_IDEA_CBC_PAD

824 **2.6.22.7.2 DES secret key objects**

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

828 *Table 20, DES Secret Key Object*

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

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

830 DES keys **must always MUST** have their parity bits properly set as described in FIPS PUB 46-3.
 831 Attempting to create or unwrap a DES key with incorrect parity **w###MUST** return an error.

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

```
833 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
834 CK_KEY_TYPE keyType = CKK_DES;
835 CK_UTF8CHAR label[] = "A DES secret key object";
836 CK_BYTE value[8] = {...};
837 CK_BBOOL true = CK_TRUE;
838 CK_ATTRIBUTE template[] = {
```



```

839     {CKA_CLASS, &class, sizeof(class)},
840     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
841     {CKA_TOKEN, &>true, sizeof(true)},
842     {CKA_LABEL, label, sizeof(label)-1},
843     {CKA_ENCRYPT, &>true, sizeof(true)},
844     {CKA_VALUE, value, sizeof(value)}
845 };

```

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

849 **2-6.32.7.3 CAST secret key objects**

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

853 *Table 21, CAST Secret Key Object Attributes*

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

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

855

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

```

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

```

870 **2-6.42.7.4 CAST3 secret key objects**

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

874 *Table 22, CAST3 Secret Key Object Attributes*

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

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

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

```

877 CK_OBJECT_CLASS class = CKO_SECRET_KEY;

```

```

878 CK_KEY_TYPE keyType = CKK_CAST3;
879 CK_UTF8CHAR label[] = "A CAST3 secret key object";
880 CK_BYTE value[] = {...};
881 CK_BBOOL true = CK_TRUE;
882 CK_ATTRIBUTE template[] = {
883     {CKA_CLASS, &class, sizeof(class)},
884     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
885     {CKA_TOKEN, &>true, sizeof(true)},
886     {CKA_LABEL, label, sizeof(label)-1},
887     {CKA_ENCRYPT, &>true, sizeof(true)},
888     {CKA_VALUE, value, sizeof(value)}
889 };

```

890 2.6.52.7.5 CAST128 (CAST5) secret key objects

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

894 *Table 23, CAST128 (CAST5) Secret Key Object Attributes*

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

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

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

```

897 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
898 CK_KEY_TYPE keyType = CKK_CAST128;
899 CK_UTF8CHAR label[] = "A CAST128 secret key object";
900 CK_BYTE value[] = {...};
901 CK_BBOOL true = CK_TRUE;
902 CK_ATTRIBUTE template[] = {
903     {CKA_CLASS, &class, sizeof(class)},
904     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
905     {CKA_TOKEN, &>true, sizeof(true)},
906     {CKA_LABEL, label, sizeof(label)-1},
907     {CKA_ENCRYPT, &>true, sizeof(true)},
908     {CKA_VALUE, value, sizeof(value)}
909 };

```

910

911 2.6.62.7.6 IDEA secret key objects

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

914 *Table 24, IDEA Secret Key Object*

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

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

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

```

917 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
918 CK_KEY_TYPE keyType = CKK_IDEA;
919 CK_UTF8CHAR label[] = "An IDEA secret key object";

```

```

920 CK_BYTE value[16] = {...};
921 CK_BBOOL true = CK_TRUE;
922 CK_ATTRIBUTE template[] = {
923     {CKA_CLASS, &class, sizeof(class)},
924     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
925     {CKA_TOKEN, &>true, sizeof(true)},
926     {CKA_LABEL, label, sizeof(label)-1},
927     {CKA_ENCRYPT, &>true, sizeof(true)},
928     {CKA_VALUE, value, sizeof(value)}
929 };

```

930

931 **2.6.72.7.7 CDMF secret key objects**

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

934 *Table 25, CDMF Secret Key Object*

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

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

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

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

```

940 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
941 CK_KEY_TYPE keyType = CKK_CDMF;
942 CK_UTF8CHAR label[] = "A CDMF secret key object";
943 CK_BYTE value[8] = {...};
944 CK_BBOOL true = CK_TRUE;
945 CK_ATTRIBUTE template[] = {
946     {CKA_CLASS, &class, sizeof(class)},
947     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
948     {CKA_TOKEN, &>true, sizeof(true)},
949     {CKA_LABEL, label, sizeof(label)-1},
950     {CKA_ENCRYPT, &>true, sizeof(true)},
951     {CKA_VALUE, value, sizeof(value)}
952 };

```

953 **2.6.82.7.8 General block cipher mechanism parameters**

954 **2.6.8.12.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR**

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

```

961 typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
962

```

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

964 **2.6.92.7.9 General block cipher key generation**

965 Cipher <NAME> has a key generation mechanism, “<NAME> key generation”, denoted by
 966 **CKM_<NAME>_KEY_GEN**.

967 This mechanism does not have a parameter.

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

971 When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB
 972 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits
 973 set properly.

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

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

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

984 **2.6.102.7.10 General block cipher ECB**

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

988 It does not have a parameter.

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

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

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

1000 *Table 26, General Block Cipher ECB: Key and Data Length*

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

			or CKA_VALUE_LEN	
--	--	--	------------------	--

1001 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1002 ~~may or may not~~**MAY** be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key
 1003 sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the
 1004 **CK_MECHANISM_INFO** structure specify the supported range of key sizes, in bytes. For the DES,
 1005 DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

2.6.112.7.11 General block cipher CBC

1006 Cipher <NAME> has a cipher-block chaining mode, “<NAME>-CBC”, denoted **CKM_<NAME>_CBC**. It is
 1007 a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping
 1008 with <NAME>.
 1009

1010 It has a parameter, an initialization vector for cipher block chaining mode. The initialization vector has the
 1011 same length as <NAME>’s blocksize.

1012 Constraints on key types and the length of data are summarized in the following table:

1013 *Table 27, General Block Cipher CBC; Key and Data Length*

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

1014 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1015 ~~may or may not~~**MAY** be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key
 1016 sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the
 1017 **CK_MECHANISM_INFO** structure specify the supported range of key sizes, in bytes. For the DES,
 1018 DES3 (triple-DES), IDEA, and CDMF ciphers, these fields are not used.

2.6.122.7.12 General block cipher CBC with PKCS padding

1020 Cipher <NAME> has a cipher-block chaining mode with PKCS padding, “<NAME>-CBC with PKCS
 1021 padding”, denoted **CKM_<NAME>_CBC_PAD**. It is a mechanism for single- and multiple-part encryption
 1022 and decryption; key wrapping; and key unwrapping with <NAME>. All ciphertext is padded with PKCS
 1023 padding.

1024 It has a parameter, an initialization vector for cipher block chaining mode. The initialization vector has the
 1025 same length as <NAME>’s blocksize.

1026 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the
 1027 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified
 1028 for the **CKA_VALUE_LEN** attribute.

1029
 1030 In addition to being able to wrap and unwrap secret keys, this mechanism ~~can~~**MAY** wrap and unwrap
 1031 RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys ~~(see~~
 1032 ~~Section ***MISSING REFERENCE*** for details).~~. The entries in the table below for data length
 1033 constraints when wrapping and unwrapping keys to not apply to wrapping and unwrapping private keys.

1034 Constraints on key types and the length of data are summarized in the following table:

1035 Table 28, General Block Cipher CBC with PKCS Padding: Key and Data Length

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

1036 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1037 **may-or-may-notMAY** be used. The CAST, CAST3 and CAST128 (CAST5) ciphers have variable key
 1038 sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the
 1039 **CK_MECHANISM_INFO** structure specify the supported range of key sizes, in bytes. For the DES,
 1040 DES3 (triple-DES), IDEA, and CDMF ciphers, these fields are not used.

1041 **2.6.132.7.13 General-length general block cipher MAC**

1042 Cipher <NAME> has a general-length MACing mode, “General-length <NAME>-MAC”, denoted
 1043 **CKM_<NAME>_MAC_GENERAL**. It is a mechanism for single-and multiple-part signatures and
 1044 verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB
 1045 113.

1046 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which specifies the size of the output.

1047 The output bytes from this mechanism are taken from the start of the final cipher block produced in the
 1048 MACing process.

1049 Constraints on key types and the length of input and output data are summarized in the following table:

1050 Table 29, General-length General Block Cipher MAC: Key and Data Length

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

1051 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1052 **may-or-may-notMAY** be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key
 1053 sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the
 1054 **CK_MECHANISM_INFO** structure specify the supported range of key sizes, in bytes. For the DES,
 1055 DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

1056 **2.6.142.7.14 General block cipher MAC**

1057 Cipher <NAME> has a MACing mechanism, “<NAME>-MAC”, denoted **CKM_<NAME>_MAC**. This
 1058 mechanism is a special case of the **CKM_<NAME>_MAC_GENERAL** mechanism described above. It
 1059 **always** produces an output of size half as large as <NAME>’s blocksize.

1060 This mechanism has no parameters.

1061 Constraints on key types and the length of data are summarized in the following table:

1062 Table 30, General Block Cipher MAC: Key and Data Length

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

C_Verify	<NAME>	Any	[blocksize/2]
----------	--------	-----	---------------

1063 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1064 ~~may or may not~~**MAY** be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key
1065 sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the
1066 **CK_MECHANISM_INFO** structure specify the supported range of key sizes, in bytes. For the DES,
1067 DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

1068 2.7.2.8 SKIPJACK

1069 2.7.2.8.1 Definitions

1070 This section defines the key type “CKK_SKIPJACK” for type CK_KEY_TYPE as used in the
1071 CKA_KEY_TYPE attribute of key objects.

1072 Mechanisms:

- 1073 CKM_SKIPJACK_KEY_GEN
- 1074 CKM_SKIPJACK_ECB64
- 1075 CKM_SKIPJACK_CBC64
- 1076 CKM_SKIPJACK_OFB64
- 1077 CKM_SKIPJACK_CFB64
- 1078 CKM_SKIPJACK_CFB32
- 1079 CKM_SKIPJACK_CFB16
- 1080 CKM_SKIPJACK_CFB8
- 1081 CKM_SKIPJACK_WRAP
- 1082 CKM_SKIPJACK_PRIVATE_WRAP
- 1083 CKM_SKIPJACK_RELAYX

1084 2.7.2.8.2 SKIPJACK secret key objects

1085 SKIPJACK secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_SKIPJACK**) holds a
1086 single-length MEK or a TEK. The following table defines the SKIPJACK secret object attributes, in
1087 addition to the common attributes defined for this object class:

1088 *Table 31, SKIPJACK Secret Key Object*

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

1089 Refer to [PKCS #11-Base] table 15 for footnotes

1090

1091 SKIPJACK keys have 16 checksum bits, and these bits must be properly set. Attempting to create or
1092 unwrap a SKIPJACK key with incorrect checksum bits **willMUST** return an error.

1093 It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK
1094 key with a specified value. Nonetheless, we provide templates for doing so.

1095 The following is a sample template for creating a SKIPJACK MEK secret key object:

```
1096 CK OBJECT CLASS class = CKO_SECRET_KEY;
1097 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1098 CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";
1099 CK_BYTE value[12] = {...};
1100 CK_BBOOL true = CK_TRUE;
1101 CK_ATTRIBUTE template[] = {
```

```

1102     {CKA_CLASS, &class, sizeof(class)},
1103     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1104     {CKA_TOKEN, &>true, sizeof(true)},
1105     {CKA_LABEL, label, sizeof(label)-1},
1106     {CKA_ENCRYPT, &>true, sizeof(true)},
1107     {CKA_VALUE, value, sizeof(value)}
1108 };

```

1109 The following is a sample template for creating a SKIPJACK TEK secret key object:

```

1110 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1111 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1112 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1113 CK_BYTE value[12] = {...};
1114 CK_BBOOL true = CK_TRUE;
1115 CK_ATTRIBUTE template[] = {
1116     {CKA_CLASS, &class, sizeof(class)},
1117     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1118     {CKA_TOKEN, &>true, sizeof(true)},
1119     {CKA_LABEL, label, sizeof(label)-1},
1120     {CKA_ENCRYPT, &>true, sizeof(true)},
1121     {CKA_WRAP, &>true, sizeof(true)},
1122     {CKA_VALUE, value, sizeof(value)}
1123 };

```

1124 2.7.3.2 **2.8.3 SKIPJACK Mechanism parameters**

1125 2.7.3.12 **2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS;** 1126 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR**

1127 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS** is a structure that provides the parameters to the
1128 **CKM_SKIPJACK_PRIVATE_WRAP** mechanism. It is defined as follows:

```

1129 typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
1130     CK_ULONG ulPasswordLen;
1131     CK_BYTE_PTR pPassword;
1132     CK_ULONG ulPublicDataLen;
1133     CK_BYTE_PTR pPublicData;
1134     CK_ULONG ulPandGLen;
1135     CK_ULONG ulQLen;
1136     CK_ULONG ulRandomLen;
1137     CK_BYTE_PTR pRandomA;
1138     CK_BYTE_PTR pPrimeP;
1139     CK_BYTE_PTR pBaseG;
1140     CK_BYTE_PTR pSubprimeQ;
1141 } CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

```

1142 The fields of the structure have the following meanings:

1143	<i>ulPasswordLen</i>	length of the password
1144	<i>pPassword</i>	pointer to the buffer which contains the user-supplied password
1145		
1146	<i>ulPublicDataLen</i>	other party's key exchange public key size
1147	<i>pPublicData</i>	pointer to other party's key exchange public key value
1148	<i>ulPandGLen</i>	length of prime and base values

1149 *ulQLen* length of subprime value

1150 *ulRandomLen* size of random Ra, in bytes

1151 *pPrimeP* pointer to Prime, p, value

1152 *pBaseG* pointer to Base, b, value

1153 *pSubprimeQ* pointer to Subprime, q, value

1154 **CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR** is a pointer to a
 1155 **CK_PRIVATE_WRAP_PARAMS**.

1156 | **2.7.3.22.8.3.2 CK_SKIPJACK_RELAYX_PARAMS;**
 1157 **CK_SKIPJACK_RELAYX_PARAMS_PTR**

1158 **CK_SKIPJACK_RELAYX_PARAMS** is a structure that provides the parameters to the
 1159 **CKM_SKIPJACK_RELAYX** mechanism. It is defined as follows:

```

1160        typedef struct CK_SKIPJACK_RELAYX_PARAMS {
1161            CK_ULONG ulOldWrappedXLen;
1162            CK_BYTE_PTR pOldWrappedX;
1163            CK_ULONG ulOldPasswordLen;
1164            CK_BYTE_PTR pOldPassword;
1165            CK_ULONG ulOldPublicDataLen;
1166            CK_BYTE_PTR pOldPublicData;
1167            CK_ULONG ulOldRandomLen;
1168            CK_BYTE_PTR pOldRandomA;
1169            CK_ULONG ulNewPasswordLen;
1170            CK_BYTE_PTR pNewPassword;
1171            CK_ULONG ulNewPublicDataLen;
1172            CK_BYTE_PTR pNewPublicData;
1173            CK_ULONG ulNewRandomLen;
1174            CK_BYTE_PTR pNewRandomA;
1175        } CK_SKIPJACK_RELAYX_PARAMS;
  
```

1176 The fields of the structure have the following meanings:

1177 *ulOldWrappedLen* length of old wrapped key in bytes

1178 *pOldWrappedX* pointer to old wrapper key

1179 *ulOldPasswordLen* length of the old password

1180 *pOldPassword* pointer to the buffer which contains the old user-supplied
 1181 password

1182 *ulOldPublicDataLen* old key exchange public key size

1183 *pOldPublicData* pointer to old key exchange public key value

1184 *ulOldRandomLen* size of old random Ra in bytes

1185 *pOldRandomA* pointer to old Ra data

1186 *ulNewPasswordLen* length of the new password

1187 *pNewPassword* pointer to the buffer which contains the new user-
 1188 supplied password

1189 *ulNewPublicDataLen* new key exchange public key size

1190 *pNewPublicData* pointer to new key exchange public key value

1191 *ulNewRandomLen* size of new random Ra in bytes

1192 *pNewRandomA* pointer to new Ra data

1193 **CK_SKIPJACK_RELAYX_PARAMS_PTR** is a pointer to a **CK_SKIPJACK_RELAYX_PARAMS**.

1194 **2.7.42.8.4 SKIPJACK key generation**

1195 The SKIPJACK key generation mechanism, denoted **CKM_SKIPJACK_KEY_GEN**, is a key generation
 1196 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1197 It does not have a parameter.

1198 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
 1199 key.

1200 **2.7.52.8.5 SKIPJACK-ECB64**

1201 SKIPJACK-ECB64, denoted **CKM_SKIPJACK_ECB64**, is a mechanism for single- and multiple-part
 1202 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB
 1203 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 cant specify a particular IV when
 1206 encrypting. It ~~can~~**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 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

1209 **2.7.62.8.6 SKIPJACK-CBC64**

1210 SKIPJACK-CBC64, denoted **CKM_SKIPJACK_CBC64**, is a mechanism for single- and multiple-part
 1211 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1212 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1213 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
 1214 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1215 Constraints on key types and the length of data are summarized in the following table:

1216 *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

1217 **2.7.72.8.7 SKIPJACK-OFB64**

1218 SKIPJACK-OFB64, denoted **CKM_SKIPJACK_OFB64**, is a mechanism for single- and multiple-part
1219 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1220 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1221 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1222 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1223 Constraints on key types and the length of data are summarized in the following table:

1224 *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

1225 **2.7.82.8.8 SKIPJACK-CFB64**

1226 SKIPJACK-CFB64, denoted **CKM_SKIPJACK_CFB64**, is a mechanism for single- and multiple-part
1227 encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.

1228 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1229 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1230 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1231 Constraints on key types and the length of data are summarized in the following table:

1232 *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

1233 **2.7.92.8.9 SKIPJACK-CFB32**

1234 SKIPJACK-CFB32, denoted **CKM_SKIPJACK_CFB32**, is a mechanism for single- and multiple-part
1235 encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.

1236 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1237 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1238 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1239 Constraints on key types and the length of data are summarized in the following table:

1240 *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

1241 **2.7.102.8.10 SKIPJACK-CFB16**

1242 SKIPJACK-CFB16, denoted **CKM_SKIPJACK_CFB16**, is a mechanism for single- and multiple-part
1243 encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.

1244 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1245 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1246 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1247 Constraints on key types and the length of data are summarized in the following table:

1248 *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

1249 2.7.112.8.11 SKIPJACK-CFB8

1250 SKIPJACK-CFB8, denoted **CKM_SKIPJACK_CFB8**, is a mechanism for single- and multiple-part
1251 encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.

1252 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1253 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1254 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1255 Constraints on key types and the length of data are summarized in the following table:

1256 *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

1257 2.7.122.8.12 SKIPJACK-WRAP

1258 The SKIPJACK-WRAP mechanism, denoted **CKM_SKIPJACK_WRAP**, is used to wrap and unwrap a
1259 secret key (MEK). It ~~can~~**MAY** wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.

1260 It does not have a parameter.

1261 2.7.132.8.13 SKIPJACK-PRIVATE-WRAP

1262 The SKIPJACK-PRIVATE-WRAP mechanism, denoted **CKM_SKIPJACK_PRIVATE_WRAP**, is used to
1263 wrap and unwrap a private key. It ~~can~~**MAY** wrap KEA and DSA private keys.

1264 It has a parameter, a **CK_SKIPJACK_PRIVATE_WRAP_PARAMS** structure.

1265 2.7.142.8.14 SKIPJACK-RELAYX

1266 The SKIPJACK-RELAYX mechanism, denoted **CKM_SKIPJACK_RELAYX**, is used with the **C_WrapKey**
1267 function to “change the wrapping” on a private key which was wrapped with the SKIPJACK-PRIVATE-
1268 WRAP mechanism (See Section 2.8.13).

1269 It has a parameter, a **CK_SKIPJACK_RELAYX_PARAMS** structure.

1270 Although the SKIPJACK-RELAYX mechanism is used with **C_WrapKey**, it differs from other key-
1271 wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to
1272 **C_WrapKey**; however for the SKIPJACK_RELAYX mechanism, the [always invalid] value 0 should be
1273 passed as the key handle for **C_WrapKey**, and the already-wrapped key should be passed in as part of
1274 the **CK_SKIPJACK_RELAYX_PARAMS** structure.

1275 2.8.22.9 BATON

1276 2.8.122.9.1 Definitions

1277 This section defines the key type “CKK_BATON” for type CK_KEY_TYPE as used in the
1278 CKA_KEY_TYPE attribute of key objects.

- 1279 Mechanisms:
- 1280 CKM_BATON_KEY_GEN
- 1281 CKM_BATON_ECB128
- 1282 CKM_BATON_ECB96
- 1283 CKM_BATON_CBC128
- 1284 CKM_BATON_COUNTER
- 1285 CKM_BATON_SHUFFLE
- 1286 CKM_BATON_WRAP

2.8.22.9.2 BATON secret key objects

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

Table 39, BATON Secret Key Object

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

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

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

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.

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

```

CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_BATON;
CK_UTF8CHAR label[] = "A BATON MEK secret key object";
CK_BYTE value[40] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &>true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &>true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};

```

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

```

CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_BATON;
CK_UTF8CHAR label[] = "A BATON TEK secret key object";
CK_BYTE value[40] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &>true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &>true, sizeof(true)},
    {CKA_WRAP, &>true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};

```

1326

```
} ;
```

1327 **2.8.32.9.3 BATON key generation**

1328 The BATON key generation mechanism, denoted **CKM_BATON_KEY_GEN**, is a key generation
1329 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

1330 It does not have a parameter.

1331 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
1332 key.

1333 **2.8.42.9.4 BATON-ECB128**

1334 BATON-ECB128, denoted **CKM_BATON_ECB128**, is a mechanism for single- and multiple-part
1335 encryption and decryption with BATON in 128-bit electronic codebook mode.

1336 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1337 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1338 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1339 Constraints on key types and the length of data are summarized in the following table:

1340 *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

1341 **2.8.52.9.5 BATON-ECB96**

1342 BATON-ECB96, denoted **CKM_BATON_ECB96**, is a mechanism for single- and multiple-part encryption
1343 and decryption with BATON in 96-bit electronic codebook mode.

1344 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1345 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1346 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1347 Constraints on key types and the length of data are summarized in the following table:

1348 *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

1349 **2.8.62.9.6 BATON-CBC128**

1350 BATON-CBC128, denoted **CKM_BATON_CBC128**, is a mechanism for single- and multiple-part
1351 encryption and decryption with BATON in 128-bit cipher-block chaining mode.

1352 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1353 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1354 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1355 Constraints on key types and the length of data are summarized in the following table:

1356 *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

1357 **2.8.72.9.7 BATON-COUNTER**

1358 BATON-COUNTER, denoted **CKM_BATON_COUNTER**, is a mechanism for single- and multiple-part
 1359 encryption and decryption with BATON in counter mode.

1360 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1361 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
 1362 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1363 Constraints on key types and the length of data are summarized in the following table:

1364 *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

1365 **2.8.82.9.8 BATON-SHUFFLE**

1366 BATON-SHUFFLE, denoted **CKM_BATON_SHUFFLE**, is a mechanism for single- and multiple-part
 1367 encryption and decryption with BATON in shuffle mode.

1368 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1369 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
 1370 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1371 Constraints on key types and the length of data are summarized in the following table:

1372 *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

1373 **2.8.92.9.9 BATON WRAP**

1374 The BATON wrap and unwrap mechanism, denoted **CKM_BATON_WRAP**, is a function used to wrap
 1375 and unwrap a secret key (MEK). It ~~can~~**MAY** wrap and unwrap SKIPJACK, BATON and JUNIPER keys.

1376 It has no parameters.

1377 When used to unwrap a key, this mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and
 1378 **CKA_VALUE** attributes to it.

1379 **2.92.10 JUNIPER**

1380 **2.9.12.10.1 Definitions**

1381 This section defines the key type “CKK_JUNIPER” for type CK_KEY_TYPE as used in the
 1382 CKA_KEY_TYPE attribute of key objects.

1383 Mechanisms:

1384 CKM_JUNIPER_KEY_GEN

1385 CKM_JUNIPER_ECB128

1386 CKM_JUNIPER_CBC128
 1387 CKM_JUNIPER_COUNTER
 1388 CKM_JUNIPER_SHUFFLE
 1389 CKM_JUNIPER_WRAP

2.9.22.10.2 JUNIPER secret key objects

1391 JUNIPER secret key objects (object class **CKO_SECRET_KEY**, key type **CKK_JUNIPER**) hold single-
 1392 length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to
 1393 the common attributes defined for this object class:

1394 *Table 45, JUNIPER Secret Key Object*

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

1395 Refer to [PKCS #11-Base] table 15 for footnotes

1396

1397 JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or
 1398 unwrap a BATON key with incorrect checksum bits ~~will~~**MUST** return an error.

1399 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key
 1400 with a specified value. Nonetheless, we provide templates for doing so.

1401 The following is a sample template for creating a JUNIPER MEK secret key object:

```

1402 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1403 CK_KEY_TYPE keyType = CKK_JUNIPER;
1404 CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";
1405 CK_BYTE value[40] = {...};
1406 CK_BBOOL true = CK_TRUE;
1407 CK_ATTRIBUTE template[] = {
1408     {CKA_CLASS, &class, sizeof(class)},
1409     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1410     {CKA_TOKEN, &>true, sizeof(true)},
1411     {CKA_LABEL, label, sizeof(label)-1},
1412     {CKA_ENCRYPT, &>true, sizeof(true)},
1413     {CKA_VALUE, value, sizeof(value)}
1414 };
  
```

1415 The following is a sample template for creating a JUNIPER TEK secret key object:

```

1416 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1417 CK_KEY_TYPE keyType = CKK_JUNIPER;
1418 CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";
1419 CK_BYTE value[40] = {...};
1420 CK_BBOOL true = CK_TRUE;
1421 CK_ATTRIBUTE template[] = {
1422     {CKA_CLASS, &class, sizeof(class)},
1423     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1424     {CKA_TOKEN, &>true, sizeof(true)},
1425     {CKA_LABEL, label, sizeof(label)-1},
1426     {CKA_ENCRYPT, &>true, sizeof(true)},
1427     {CKA_WRAP, &>true, sizeof(true)},
1428     {CKA_VALUE, value, sizeof(value)}
1429 };
  
```

2.9.32.10.3 JUNIPER key generation

1431 The JUNIPER key generation mechanism, denoted **CKM_JUNIPER_KEY_GEN**, is a key generation
 1432 mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).

1433 It does not have a parameter.
 1434 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
 1435 key.

1436 **2.9.42.10.4 JUNIPER-ECB128**

1437 JUNIPER-ECB128, denoted **CKM_JUNIPER_ECB128**, is a mechanism for single- and multiple-part
 1438 encryption and decryption with JUNIPER in 128-bit electronic codebook mode.
 1439 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1440 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
 1441 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.
 1442 Constraints on key types and the length of data are summarized in the following table. For encryption
 1443 and decryption, the input and output data (parts) ~~may~~**MAY** begin at the same location in memory.

1444 *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

1445 **2.9.52.10.5 JUNIPER-CBC128**

1446 JUNIPER-CBC128, denoted **CKM_JUNIPER_CBC128**, is a mechanism for single- and multiple-part
 1447 encryption and decryption with JUNIPER in 128-bit cipher block chaining mode.
 1448 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1449 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
 1450 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.
 1451 Constraints on key types and the length of data are summarized in the following table. For encryption
 1452 and decryption, the input and output data (parts) ~~may~~**MAY** begin at the same location in memory.

1453 *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

1454 **2.9.62.10.6 JUNIPER-COUNTER**

1455 JUNIPER-COUNTER, denoted **CKM_JUNIPER_COUNTER**, is a mechanism for single- and multiple-
 1456 part encryption and decryption with JUNIPER in counter mode.
 1457 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
 1458 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
 1459 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.
 1460 Constraints on key types and the length of data are summarized in the following table. For encryption
 1461 and decryption, the input and output data (parts) ~~may~~**MAY** begin at the same location in memory.

1462 *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

1463 **2.9.72.10.7 JUNIPER-SHUFFLE**

1464 JUNIPER-SHUFFLE, denoted **CKM_JUNIPER_SHUFFLE**, is a mechanism for single- and multiple-part
1465 encryption and decryption with JUNIPER in shuffle mode.

1466 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some
1467 value generated by the token – in other words, the application ~~cannot~~**MAY NOT** specify a particular IV
1468 when encrypting. It ~~can~~**MAY**, of course, specify a particular IV when decrypting.

1469 Constraints on key types and the length of data are summarized in the following table. For encryption
1470 and decryption, the input and output data (parts) ~~may~~**MAY** begin at the same location in memory.

1471 *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

1472 **2.9.82.10.8 JUNIPER WRAP**

1473 The JUNIPER wrap and unwrap mechanism, denoted **CKM_JUNIPER_WRAP**, is a function used to wrap
1474 and unwrap an MEK. It ~~can~~**MAY** wrap or unwrap SKIPJACK, BATON and JUNIPER keys.

1475 It has no parameters.

1476 When used to unwrap a key, this mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and
1477 **CKA_VALUE** attributes to it.

1478 **2.102.11 MD2**

1479 **2.10.12.11.1 Definitions**

1480 Mechanisms:

1481 CKM_MD2

1482 CKM_MD2_HMAC

1483 CKM_MD2_HMAC_GENERAL

1484 CKM_MD2_KEY_DERIVATION

1485 **2.10.22.11.2 MD2 digest**

1486 The MD2 mechanism, denoted **CKM_MD2**, is a mechanism for message digesting, following the MD2
1487 message-digest algorithm defined in RFC ~~6134~~49.

1488 It does not have a parameter.

1489 Constraints on the length of data are summarized in the following table:

1490 *Table 50, MD2: Data Length*

Function	Data length	Digest Length
C_Digest	Any	16

1491 **2.10.32.11.3 General-length MD2-HMAC**

1492 The general-length MD2-HMAC mechanism, denoted **CKM_MD2_HMAC_GENERAL**, is a mechanism for
1493 signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it
1494 uses are generic secret keys.

1495 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1496 output. This length should be in the range 0-16 (the output size of MD2 is 16 bytes). Signatures (MACs)
1497 produced by this mechanism **willMUST** be taken from the start of the full 16-byte HMAC output.

1498 *Table 51, General-length MD2-HMAC: Key and Data Length*

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

1499 **2.10.42.11.4 MD2-HMAC**

1500 The MD2-HMAC mechanism, denoted **CKM_MD2_HMAC**, is a special case of the general-length MD2-
1501 HMAC mechanism in Section 2.11.3.

1502 It has no parameter, and **always** produces an output of length 16.

1503 **2.10.52.11.5 MD2 key derivation**

1504 MD2 key derivation, denoted **CKM_MD2_KEY_DERIVATION**, is a mechanism which provides the
1505 capability of deriving a secret key by digesting the value of another secret key with MD2.

1506 The value of the base key is digested once, and the result is used to make the value of the derived secret
1507 key.

- 1508 • If no length or key type is provided in the template, then the key produced by this mechanism
1509 **willMUST** be a generic secret key. Its length **willMUST** be 16 bytes (the output size of MD2)..
 - 1510 • If no key type is provided in the template, but a length is, then the key produced by this mechanism
1511 **willMUST** be a generic secret key of the specified length.
 - 1512 • ~~If no length was provided in the template, but a key type is, then that key type must have a well-~~
1513 ~~defined length. If it does, then the key produced by this mechanism. If no length was provided in the~~
1514 ~~template, but a key type is, then that key type must have a well-defined length. If it does, then the~~
1515 ~~key produced by this mechanism. willMUST~~ be of the type specified in the template. If it doesn't, an
1516 error **willMUST** be returned.
 - 1517 • If both a key type and a length are provided in the template, the length must be compatible with that
1518 key type. The key produced by this mechanism **willMUST** be of the specified type and length.
- 1519 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key **willMUST** be set
1520 properly.
- 1521 If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.
- 1522 This mechanism has the following rules about key sensitivity and extractability:
- 1523 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key ~~can~~**MAY**
1524 both be specified to be either CK_TRUE or CK_FALSE. If omitted, these attributes each take on
1525 some default value.
 - 1526 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, then the derived key
1527 **willMUST** as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_TRUE,
1528 then the derived key has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value as its
1529 **CKA_SENSITIVE** attribute.
 - 1530 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE, then the
1531 derived key **willMUST**, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to
1532 CK_TRUE, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the *opposite*
1533 value from its **CKA_EXTRACTABLE** attribute.

1534 **2.11.2.12 MD5**

1535 **2.11.2.12.1 Definitions**

1536 Mechanisms:

- 1537 CKM_MD5
- 1538 CKM_MD5_HMAC
- 1539 CKM_MD5_HMAC_GENERAL
- 1540 CKM_MD5_KEY_DERIVATION

1541 **2.11.2.12.2 MD5 Digest**

1542 The MD5 mechanism, denoted **CKM_MD5**, is a mechanism for message digesting, following the MD5
1543 message-digest algorithm defined in RFC 1321.

1544 It does not have a parameter.

1545 Constraints on the length of input and output data are summarized in the following table. For single-part
1546 digesting, the data and the digest ~~may~~**MAY** begin at the same location in memory.

1547 *Table 52, MD5: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1548 **2.11.3.12.3 General-length MD5-HMAC**

1549 The general-length MD5-HMAC mechanism, denoted **CKM_MD5_HMAC_GENERAL**, is a mechanism for
1550 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it
1551 uses are generic secret keys.

1552 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1553 output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)
1554 produced by this mechanism ~~will~~**MUST** be taken from the start of the full 16-byte HMAC output.

1555 *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

1556 **2.11.4.12.4 MD5-HMAC**

1557 The MD5-HMAC mechanism, denoted **CKM_MD5_HMAC**, is a special case of the general-length MD5-
1558 HMAC mechanism in Section 2.12.3.

1559 It has no parameter, and ~~always~~ produces an output of length 16.

1560 **2.11.5.12.5 MD5 key derivation**

1561 MD5 key derivation denoted **CKM_MD5_KEY_DERIVATION**, is a mechanism which provides the
1562 capability of deriving a secret key by digesting the value of another secret key with MD5.

1563 The value of the base key is digested once, and the result is used to make the value of derived secret
1564 key.

- 1565 • If no length or key type is provided in the template, then the key produced by this mechanism
1566 ~~will~~**MUST** be a generic secret key. Its length ~~will~~**MUST** be 16 bytes (the output size of MD5).

- 1567 • If no key type is provided in the template, but a length is, then the key produced by this mechanism
1568 ~~will~~**MUST** be a generic secret key of the specified length.
- 1569 • ~~If no length was provided in the template, but a key type is, then that key type must have a well-~~
1570 ~~defined length. If it does, then the key produced by this mechanism **MUST** if no length was provided~~
1571 ~~in the template, but a key type is, then that key type must have a well-defined length. If it does, then~~
1572 ~~the key produced by this mechanism ~~will~~~~ be of the type specified in the template. If it doesn't, an
1573 error ~~will~~**MUST** be returned.
- 1574 • If both a key type and a length are provided in the template, the length must be compatible with that
1575 key type. The key produced by this mechanism ~~will~~**MUST** be of the specified type and length.
- 1576 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key ~~will~~**MUST** be set
1577 properly.
- 1578 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.
- 1579 This mechanism has the following rules about key sensitivity and extractability.
- 1580 • The **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes in the template for the new key ~~can~~**MAY**
1581 both be specified to either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some
1582 default value.
- 1583 • If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, then the derived key
1584 ~~will~~**MUST** as well. If the base key has its **CKA_ALWAYS_SENSITIVE** attribute set to CK_TRUE,
1585 then the derived key has its **CKA_ALWAYS_SENSITIVE** attribute set to the same value as its
1586 **CKA_SENSITIVE** attribute.
- 1587 • Similarly, if the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE, then the
1588 derived key ~~will~~**MUST**, too. If the base key has its **CKA_NEVER_EXTRACTABLE** attribute set to
1589 CK_TRUE, then the derived key has its **CKA_NEVER_EXTRACTABLE** attribute set to the *opposite*
1590 value from its **CKA_EXTRACTABLE** attribute.

1591 2.122.13 FASTHASH

1592 2.12.12.13.1 Definitions

1593 Mechanisms:

1594 CKM_FASTHASH

1595 2.12.22.13.2 FASTHASH digest

1596 The FASTHASH mechanism, denoted **CKM_FASTHASH**, is a mechanism for message digesting,
1597 following the U.S. government's algorithm.

1598 It does not have a parameter.

1599 Constraints on the length of input and output data are summarized in the following table:

1600 *Table 54, FASTHASH: Data Length*

Function	Input length	Digest length
C_Digest	Any	40

1601

1602 2.132.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)

1603 2.14.1 Definitions

1604 The mechanisms in this section are for generating keys and IVs for performing password-based
1605 encryption. The method used to generate keys and IVs is specified in PKCS #5.

1606 **2.13.11.1.1 Definitions**

1607 Mechanisms:

1608 CKM_PBE_MD2_DES_CBC
1609 CKM_PBE_MD5_DES_CBC
1610 CKM_PBE_MD5_CAST_CBC
1611 CKM_PBE_MD5_CAST3_CBC
1612 CKM_PBE_MD5_CAST5_CBC
1613 CKM_PBE_MD5_CAST128_CBC
1614 CKM_PBE_SHA1_CAST5_CBC
1615 CKM_PBE_SHA1_CAST128_CBC
1616 CKM_PBE_SHA1_RC4_128
1617 CKM_PBE_SHA1_RC4_40
1618 CKM_PBE_SHA1_RC2_128_CBC
1619 CKM_PBE_SHA1_RC2_40_CBC

1620 **2.13.22.14.2 Password-based encryption/authentication mechanism** 1621 **parameters**

1622 **2.13.2.12.14.2.1 CK_PBE_PARAMS; CK_PBE_PARAMS_PTR**

1623 **CK_PBE_PARAMS** is a structure which provides all of the necessary information required by the
1624 CKM_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation
1625 mechanisms) and the CKM_PBA_SHA1_WITH_SHA1_HMAC mechanism. It is defined as follows:

```
1626 typedef struct CK_PBE_PARAMS {  
1627     CK_BYTE_PTR pInitVector;  
1628     CK_UTF8CHAR_PTR pPassword;  
1629     CK_ULONG ulPasswordLen;  
1630     CK_BYTE_PTR pSalt;  
1631     CK_ULONG ulSaltLen;  
1632     CK_ULONG ulIteration;  
1633 } CK_PBE_PARAMS;
```

1634 The fields of the structure have the following meanings:

1635	<i>pInitVector</i>	pointer to the location that receives the 8-byte
1636		initialization vector (IV), if an IV is required
1637	<i>pPassword</i>	points to the password to be used in the PBE key
1638		generation
1639	<i>ulPasswordLen</i>	length in bytes of the password information
1640	<i>pSalt</i>	points to the salt to be used in the PBE key generation
1641	<i>ulSaltLen</i>	length in bytes of the salt information
1642	<i>ullteration</i>	number of iterations required for the generation

1643 **CK_PBE_PARAMS_PTR** is a pointer to a **CK_PBE_PARAMS**.

1644 | **2.13.32.14.3 MD2-PBE for DES-CBC**

1645 | MD2-PBE for DES-CBC, denoted **CKM_PBE_MD2_DES_CBC**, is a mechanism used for generating a
1646 | DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an
1647 | iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1648 | It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1649 | key generation process and the location of the application-supplied buffer which will receiverreceives the
1650 | 8-byte IV generated by the mechanism.

1651 | **2.13.42.14.4 MD5-PBE for DES-CBC**

1652 | MD5-PBE for DES-CBC, denoted **CKM_PBE_MD5_DES_CBC**, is a mechanism used for generating a
1653 | DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
1654 | iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1655 | It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1656 | key generation process and the location of the application-supplied buffer which will receiverreceives the
1657 | 8-byte IV generated by the mechanism.

1658 | **2.13.52.14.5 MD5-PBE for CAST-CBC**

1659 | MD5-PBE for CAST-CBC, denoted **CKM_PBE_MD5_CAST_CBC**, is a mechanism used for generating a
1660 | CAST secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an
1661 | iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1662 | It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1663 | key generation process and the location of the application-supplied buffer which will receiverreceives the
1664 | 8-byte IV generated by the mechanism

1665 | The length of the CAST key generated by this mechanism mayMAY be specified in the supplied template;
1666 | if it is not present in the template, it defaults to 8 bytes.

1667 | **2.13.62.14.6 MD5-PBE for CAST3-CBC**

1668 | MD5-PBE for CAST3-CBC, denoted **CKM_PBE_MD5_CAST3_CBC**, is a mechanism used for generating
1669 | a CAST3 secret key and an IV from a password and a salt value by using the MD5 digest algorithm and
1670 | an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1671 | It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1672 | key generation process and the location of the application-supplied buffer which will receiverreceives the
1673 | 8-byte IV generated by the mechanism

1674 | The length of the CAST3 key generated by this mechanism mayMAY be specified in the supplied
1675 | template; if it is not present in the template, it defaults to 8 bytes.

1676 | **2.13.72.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC)**

1677 | MD5-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM_PBE_MD5_CAST128_CBC** or
1678 | **CKM_PBE_MD5_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
1679 | and an IV from a password and a salt value by using the MD5 digest algorithm and an iteration count.
1680 | This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1681 | It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1682 | key generation process and the location of the application-supplied buffer which will receiverreceives the
1683 | 8-byte IV generated by the mechanism

1684 | The length of the CAST128 (CAST5) key generated by this mechanism mayMAY be specified in the
1685 | supplied template; if it is not present in the template, it defaults to 8 bytes.

1686 | **2.13.82.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)**

1687 | SHA-1-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM_PBE_SHA1_CAST128_CBC** or
1688 | **CKM_PBE_SHA1_CAST5_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key
1689 | and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This
1690 | functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1691 | It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1692 | key generation process and the location of the application-supplied buffer which ~~will receive~~receives the
1693 | 8-byte IV generated by the mechanism

1694 | The length of the CAST128 (CAST5) key generated by this mechanism ~~may~~MAY be specified in the
1695 | supplied template; if it is not present in the template, it defaults to 8 bytes

1696 | **2.142.15 PKCS #12 password-based encryption/authentication**
1697 | **mechanisms**

1698 | **2.15.1 Definitions**

1699 | The mechanisms in this section are for generating keys and IVs for performing password-based
1700 | encryption or authentication. The method used to generate keys and IVs is based on a method that was
1701 | specified in PKCS #12.

1702 | We specify here a general method for producing various types of pseudo-random bits from a password,
1703 | p ; a string of salt bits, s ; and an iteration count, c . The “type” of pseudo-random bits to be produced is
1704 | identified by an identification byte, ID , described at the ~~meaningend~~ of ~~which will be discussed later~~this
1705 | section.

1706 | Let H be a hash function built around a compression function $f: \mathbf{Z}_2^u \times \mathbf{Z}_2^v \rightarrow \mathbf{Z}_2^u$ (that is, H has a chaining
1707 | variable and output of length u bits, and the message input to the compression function of H is v bits). For
1708 | MD2 and MD5, $u=128$ and $v=512$; for SHA-1, $u=160$ and $v=512$.

1709 | We assume here that u and v are both multiples of 8, as are the lengths in bits of the password and salt
1710 | strings and the number n of pseudo-random bits required. In addition, u and v are of course nonzero.

- 1711 | 1. Construct a string, D (the “diversifier”), by concatenating $v/8$ copies of ID .
- 1712 | 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
1713 | the salt ~~may~~MAY be truncated to create S). Note that if the salt is the empty string, then so is S
- 1714 | 3. Concatenate copies of the password together to create a string P of length $v \cdot \lceil p/v \rceil$ bits (the final
1715 | copy of the password ~~may~~MAY be truncated to create P). Note that if the password is the empty
1716 | string, then so is P .
- 1717 | 4. Set $I = S || P$ to be the concatenation of S and P .
- 1718 | 5. Set $j = \lceil n/u \rceil$.
- 1719 | 6. For $i = 1, 2, \dots, j$, do the following:
 - 1720 | a. Set $A_i = H_c(D || I)$, the i th hash of $D || I$. That is, compute the hash of $D || I$; compute the hash
1721 | of that hash; etc.; continue in this fashion until a total of c hashes have been computed,
1722 | each on the result of the previous hash.
 - 1723 | b. Concatenate copies of A_i to create a string B of length v bits (the final copy of A_i ~~may~~MAY
1724 | be truncated to create B).
 - 1725 | c. Treating I as a concatenation I_0, I_1, \dots, I_{k-1} of v -bit blocks, where $k = \lceil s/v \rceil + \lceil p/v \rceil$, modify I
1726 | by setting $I_j = (I_j + B + 1) \bmod 2^v$ for each j . To perform this addition, treat each v -bit block as
1727 | a binary number represented most-significant bit first
- 1728 | 7. Concatenate A_1, A_2, \dots, A_j together to form a pseudo-random bit string, A .
- 1729 | 8. Use the first n bits of A as the output of this entire process

1730 | When the password-based encryption mechanisms presented in this section are used to generate a key
1731 | and IV (if needed) from a password, salt, and an iteration count, the above algorithm is used. To

1732 generate a key, the identifier byte *ID* is set to the value 1; to generate an IV, the identifier byte *ID* is set to
1733 the value 2.

1734 When the password-based authentication mechanism presented in this section is used to generate a key
1735 from a password, salt and an iteration count, the above algorithm is used. The identifier *ID* is set to the
1736 value 3.

1737 **2.14.12.15.2 SHA-1-PBE for 128-bit RC4**

1738 SHA-1-PBE for 128-bit RC4, denoted **CKM_PBE_SHA1_RC4_128**, is a mechanism used for generating
1739 a 128-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
1740 iteration count. The method used to generate the key is described above.

1741 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1742 key generation process. The parameter also has a field to hold the location of an application-supplied
1743 buffer which **will receiverreceives** an IV; for this mechanism, the contents of this field are ignored, since
1744 RC4 does not require an IV.

1745 The key produced by this mechanism will typically be used for performing password-based encryption.

1746 **2.14.22.15.3 SHA-1_PBE for 40-bit RC4**

1747 SHA-1-PBE for 40-bit RC4, denoted **CKM_PBE_SHA1_RC4_40**, is a mechanism used for generating a
1748 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an
1749 iteration count. The method used to generate the key is described above.

1750 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1751 key generation process. The parameter also has a field to hold the location of an application-supplied
1752 buffer which **will receiverreceives** an IV; for this mechanism, the contents of this field are ignored, since
1753 RC4 does not require an IV.

1754 The key produced by this mechanism will typically be used for performing password-based encryption.

1755 **2.14.32.15.4 SHA-1_PBE for 128-bit RC2-CBC**

1756 SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_128_CBC**, is a mechanism used for
1757 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest
1758 algorithm and an iteration count. The method used to generate the key and IV is described above.

1759 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1760 key generation process and the location of an application-supplied buffer which **will receiverreceives** the 8-
1761 byte IV generated by the mechanism.

1762 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
1763 of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object
1764 Identifier `pbeWithSHA1And128BitRC2-CBC`.

1765 The key and IV produced by this mechanism will typically be used for performing password-based
1766 encryption.

1767 **2.14.42.15.5 SHA-1_PBE for 40-bit RC2-CBC**

1768 SHA-1-PBE for 40-bit RC2-CBC, denoted **CKM_PBE_SHA1_RC2_40_CBC**, is a mechanism used for
1769 generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm
1770 and an iteration count. The method used to generate the key and IV is described above.

1771 It has a parameter, a **CK_PBE_PARAMS** structure. The parameter specifies the input information for the
1772 key generation process and the location of an application-supplied buffer which **will receiverreceives** the 8-
1773 byte IV generated by the mechanism.

1774 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number
1775 of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object
1776 Identifier `pbeWithSHA1And40BitRC2-CBC`.

1777 The key and IV produced by this mechanism will typically be used for performing password-based
1778 encryption

1779 **2.15.2.16 RIPE-MD**

1780 **2.15.12.16.1 Definitions**

1781 Mechanisms:

1782 CKM_RIPEMD128
1783 CKM_RIPEMD128_HMAC
1784 CKM_RIPEMD128_HMAC_GENERAL
1785 CKM_RIPEMD160
1786 CKM_RIPEMD160_HMAC
1787 CKM_RIPEMD160_HMAC_GENERAL

1788 **2.15.22.16.2 RIPE-MD 128 Digest**

1789 The RIPE-MD 128 mechanism, denoted **CKM_RIPEMD128**, is a mechanism for message digesting,
1790 following the RIPE-MD 128 message-digest algorithm.

1791 It does not have a parameter.

1792 Constraints on the length of data are summarized in the following table:

1793 *Table 55, RIPE-MD 128: Data Length*

Function	Data length	Digest length
----------	-------------	---------------

C_Digest	Any	16
----------	-----	----

1794

1795 **2.15.32.16.3 General-length RIPE-MD 128-HMAC**

1796 The general-length RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC_GENERAL**, is
1797 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 128
1798 hash function. The keys it uses are generic secret keys.

1799 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1800 output. This length should be in the range 0-16 (the output size of RIPE-MD 128 is 16 bytes). Signatures
1801 (MACs) produced by this mechanism **will MUST** be taken from the start of the full 16-byte HMAC output.

1802 *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
----------	----------------	-----	-------------------------------

1803 **2.15.42.16.4 RIPE-MD 128-HMAC**

1804 The RIPE-MD 128-HMAC mechanism, denoted **CKM_RIPEMD128_HMAC**, is a special case of the
1805 general-length RIPE-MD 128-HMAC mechanism in Section 2.16.3.

1806 It has no parameter, and **always** produces an output of length 16.

1807 **2.15.52.16.5 RIPE-MD 160**

1808 The RIPE-MD 160 mechanism, denoted **CKM_RIPEMD160**, is a mechanism for message digesting,
1809 following the RIPE-MD 160 message-digest defined in ISO-10118.

1810 It does not have a parameter.

1811 Constraints on the length of data are summarized in the following table:

1812 *Table 57, RIPE-MD 160: Data Length*

Function	Data length	Digest length
C_Digest	Any	20

1813 **2.15.62.16.6 General-length RIPE-MD 160-HMAC**

1814 The general-length RIPE-MD 160-HMAC mechanism, denoted **CKM_RIPEMD160_HMAC_GENERAL**, is
1815 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160
1816 hash function. The keys it uses are generic secret keys.

1817 It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired
1818 output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures
1819 (MACs) produced by this mechanism **willMUST** be taken from the start of the full 20-byte HMAC output.

1820 *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

1821 **2.15.72.16.7 RIPE-MD 160-HMAC**

1822 The RIPE-MD 160-HMAC mechanism, denoted **CKM_RIPEMD160_HMAC**, is a special case of the
1823 general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.

1824 It has no parameter, and **always** produces an output of length 20.

1825 **2.162.17 SET**

1826 **2.16.12.17.1 Definitions**

1827 Mechanisms:

1828 **CKM_KEY_WRAP_SET_OAEP**

1829 **2.16.22.17.2 SET mechanism parameters**

1830 **2.16.2.12.17.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS;** 1831 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR**

1832 **CK_KEY_WRAP_SET_OAEP_PARAMS** is a structure that provides the parameters to the
1833 **CKM_KEY_WRAP_SET_OAEP** mechanism. It is defined as follows:

```
1834 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {  
1835     CK_BYTE bbc;  
1836     CK_BYTE_PTR pX;  
1837     CK_ULONG ulXLen;  
1838 } CK_KEY_WRAP_SET_OAEP_PARAMS;
```

1839 The fields of the structure have the following meanings:

1840 *bBC* block contents byte

1841 *pX* concatenation of hash of plaintext data (if present) and
1842 extra data (if present)

1843 *ulXLen* length in bytes of concatenation of hash of plaintext data
1844 (if present) and extra data (if present). 0 if neither is
1845 present.

1846 **CK_KEY_WRAP_SET_OAEP_PARAMS_PTR** is a pointer to a
1847 **CK_KEY_WRAP_SET_OAEP_PARAMS**.

1848 **2.16.32.17.3 OAEP key wrapping for SET**

1849 The OAEP key wrapping for SET mechanism, denoted **CKM_KEY_WRAP_SET_OAEP**, is a mechanism
1850 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some
1851 extra data ~~may optionally~~**MAY** be wrapped together with the DES key. This mechanism is defined in the
1852 SET protocol specifications.

1853 It takes a parameter, a **CK_KEY_WRAP_SET_OAEP_PARAMS** structure. This structure holds the
1854 “Block Contents” byte of the data and the concatenation of the hash of plaintext data (if present) and the
1855 extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by
1856 the *ulXLen* field having the value 0.

1857 When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present)
1858 and the extra data (if present) is returned following the convention described ~~in Section ***MISSING~~
1859 ~~REFERENCE*** on producing output.~~**[PKCS #11-Curr], Miscellaneous simple key derivation**
1860 **mechanisms**. Note that if the inputs to **C_UnwrapKey** are such that the extra data is not returned (e.g.
1861 the buffer supplied in the **CK_KEY_WRAP_SET_OAEP_PARAMS** structure is **NULL_PTR**), then the
1862 unwrapped key object ~~will not~~**MUST NOT** be created, either.

1863 Be aware that when this mechanism is used to unwrap a key, the *bBC* and *pX* fields of the parameter
1864 supplied to the mechanism ~~may~~**MAY** be modified.

1865 If an application uses **C_UnwrapKey** with **CKM_KEY_WRAP_SET_OAEP**, it may be preferable for it
1866 simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data
1867 (this concatenation ~~is never~~ **MUST NOT be** larger than 128 bytes), rather than calling **C_UnwrapKey**
1868 twice. Each call of **C_UnwrapKey** with **CKM_KEY_WRAP_SET_OAEP** requires an RSA decryption
1869 operation to be performed, and this computational overhead ~~can~~**MAY** be avoided by this means.

1870 **2.172.18 LYNKS**

1871 **2.17.12.18.1 Definitions**

1872 Mechanisms:

1873 **CKM_KEY_WRAP_LYNKS**

1874 **2.17.22.18.2 LYNKS key wrapping**

1875 The LYNKS key wrapping mechanism, denoted **CKM_KEY_WRAP_LYNKS**, is a mechanism for
1876 wrapping and unwrapping secret keys with DES keys. It ~~can~~**MAY** wrap any 8-byte secret key, and it
1877 produces a 10-byte wrapped key, containing a cryptographic checksum.

1878 It does not have a parameter.

1879 To wrap an 8-byte secret key *K* with a DES key *W*, this mechanism performs the following steps:

- 1880 1. Initialize two 16-bit integers, *sum*₁ and *sum*₂, to 0
- 1881 2. Loop through the bytes of *K* from first to last.

- 1882 3. Set $sum_1 = sum_1 + \text{the key byte}$ (treat the key byte as a number in the range 0-255).
1883 4. Set $sum_2 = sum_2 + sum_1$.
1884 5. Encrypt K with W in ECB mode, obtaining an encrypted key, E .
1885 6. Concatenate the last 6 bytes of E with sum_2 , representing sum_2 most-significant bit first. The
1886 result is an 8-byte block, T .
1887 7. Encrypt T with W in ECB mode, obtaining an encrypted checksum, C .
1888 8. Concatenate E with the last 2 bytes of C to obtain the wrapped key.

1889 When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly,
1890 an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity
1891 bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.

1892

1893 **3 PKCS #11 Implementation Conformance**

1894 An implementation is a conforming implementation if it meets the conditions specified in one or more
1895 server profiles specified in **[PKCS #11-Prof]**.

1896 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1897 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL
1898 conform to all normative statements within the clauses specified for that profile and for any subclauses to
1899 each of those clauses .

1900

1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940

Appendix A. Acknowledgments

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

Participants:

- Gil Abel, Athena Smartcard Solutions, Inc.
- Warren Armstrong, QuintessenceLabs
- Peter Bartok, Venafi, Inc.
- Anthony Berglas, Cryptsoft
- Kelley Burgin, National Security Agency
- Robert Burns, Thales e-Security
- Wan-Teh Chang, Google Inc.
- Hai-May Chao, Oracle
- Janice Cheng, Vormetric, Inc.
- Sangrae Cho, Electronics and Telecommunications Research Institute (ETRI)
- Doron Cohen, SafeNet, Inc.
- Fadi Cotran, Futurex
- Tony Cox, Cryptsoft
- Christopher Duane, EMC
- Chris Dunn, SafeNet, Inc.
- Valerie Fenwick, Oracle
- Terry Fletcher, SafeNet, Inc.
- Susan Gleeson, Oracle
- Sven Gossel, Charismathics
- Robert Griffin, EMC
- Paul Grojean, Individual
- Peter Gutmann, Individual
- Dennis E. Hamilton, Individual
- Thomas Hardjono, M.I.T.
- Tim Hudson, Cryptsoft
- Gershon Janssen, Individual
- Seunghun Jin, Electronics and Telecommunications Research Institute (ETRI)
- Andrey Jivsov, Symantec Corp.
- Greg Kazmierczak, Wave Systems Corp.
- Mark Knight, Thales e-Security
- Darren Krahn, Google Inc.
- Alex Krasnov, Infineon Technologies AG
- Dina Kurktchi-Nimeh, Oracle
- Mark Lambiase, SecureAuth Corporation
- Lawrence Lee, GoTrust Technology Inc.

1941 John Leiseboer, QuintessenceLabs
1942 Hal Lockhart, Oracle
1943 Robert Lockhart, Thales e-Security
1944 Dale Moberg, Axway Software
1945 Darren Moffat, Oracle
1946 Valery Osheter, SafeNet, Inc.
1947 Sean Parkinson, EMC
1948 Rob Philpott, EMC
1949 Mark Powers, Oracle
1950 Ajai Puri, SafeNet, Inc.
1951 Robert Relyea, Red Hat
1952 Saikat Saha, Oracle
1953 Subhash Sankuratipati, NetApp
1954 Johann Schoetz, Infineon Technologies AG
1955 Rayees Shamsuddin, Wave Systems Corp.
1956 Radhika Siravara, Oracle
1957 Brian Smith, Mozilla Corporation
1958 David Smith, Venafi, Inc.
1959 Ryan Smith, Futurex
1960 Jerry Smith, US Department of Defense (DoD)
1961 Oscar So, Oracle
1962 Michael Stevens, QuintessenceLabs
1963 Michael StJohns, Individual
1964 Sander Temme, Thales e-Security
1965 Kiran Thota, VMware, Inc.
1966 Walter-John Turnes, Gemini Security Solutions, Inc.
1967 Stef Walter, Red Hat
1968 Jeff Webb, Dell
1969 Magda Zdunkiewicz, Cryptsoft
1970 Chris Zimman, Bloomberg Finance L.P.

1971

Appendix B. Manifest constants

1972 The following constants have been defined for PKCS #11 V2.40. Also, refer to **[PKCS #11-Base]** and
1973 **[PKCS #11-Curr]** for additional definitions.

```
1974 /*  
1975 * Copyright OASIS Open 20134. All rights reserved.  
1976 * OASIS trademark, IPR and other policies apply.  
1977 * http://www.oasis-open.org/policies-guidelines/ipr  
1978 */  
1979  
1980 #define CKK_KEA 0x00000005  
1981 #define CKK_RC2 0x00000011  
1982 #define CKK_RC4 0x00000012  
1983 #define CKK_DES 0x00000013  
1984 #define CKK_CAST 0x00000016  
1985 #define CKK_CAST3 0x00000017  
1986 #define CKK_CAST5 0x00000018  
1987 #define CKK_CAST128 0x00000018  
1988 #define CKK_RC5 0x00000019  
1989 #define CKK_IDEA 0x0000001A  
1990 #define CKK_SKIPJACK 0x0000001B  
1991 #define CKK_BATON 0x0000001C  
1992 #define CKK_JUNIPER 0x0000001D  
1993 #define CKM_MD2_RSA_PKCS 0x00000004  
1994 #define CKM_MD5_RSA_PKCS 0x00000005  
1995 #define CKM_RIPEMD128_RSA_PKCS 0x00000007  
1996 #define CKM_RIPEMD160_RSA_PKCS 0x00000008  
1997 #define CKM_RC2_KEY_GEN 0x00000100  
1998 #define CKM_RC2_ECB 0x00000101  
1999 #define CKM_RC2_CBC 0x00000102  
2000 #define CKM_RC2_MAC 0x00000103  
2001 #define CKM_RC2_MAC_GENERAL 0x00000104  
2002 #define CKM_RC2_CBC_PAD 0x00000105  
2003 #define CKM_RC4_KEY_GEN 0x00000110  
2004 #define CKM_RC4 0x00000111  
2005 #define CKM_DES_KEY_GEN 0x00000120  
2006 #define CKM_DES_ECB 0x00000121  
2007 #define CKM_DES_CBC 0x00000122  
2008 #define CKM_DES_MAC 0x00000123  
2009 #define CKM_DES_MAC_GENERAL 0x00000124  
2010 #define CKM_DES_CBC_PAD 0x00000125  
2011 #define CKM_MD2 0x00000200  
2012 #define CKM_MD2_HMAC 0x00000201  
2013 #define CKM_MD2_HMAC_GENERAL 0x00000202  
2014 #define CKM_MD5 0x00000210  
2015 #define CKM_MD5_HMAC 0x00000211  
2016 #define CKM_MD5_HMAC_GENERAL 0x00000212  
2017 #define CKM_RIPEMD128 0x00000230  
2018 #define CKM_RIPEMD128_HMAC 0x00000231  
2019 #define CKM_RIPEMD128_HMAC_GENERAL 0x00000232  
2020 #define CKM_RIPEMD160 0x00000240  
2021 #define CKM_RIPEMD160_HMAC 0x00000241  
2022 #define CKM_RIPEMD160_HMAC_GENERAL 0x00000242  
2023 #define CKM_CAST_KEY_GEN 0x00000300  
2024 #define CKM_CAST_ECB 0x00000301  
2025 #define CKM_CAST_CBC 0x00000302  
2026 #define CKM_CAST_MAC 0x00000303  
2027 #define CKM_CAST_MAC_GENERAL 0x00000304  
2028 #define CKM_CAST_CBC_PAD 0x00000305  
2029 #define CKM_CAST3_KEY_GEN 0x00000310
```

```

2030 #define CKM_CAST3_ECB 0x00000311
2031 #define CKM_CAST3_CBC 0x00000312
2032 #define CKM_CAST3_MAC 0x00000313
2033 #define CKM_CAST3_MAC_GENERAL 0x00000314
2034 #define CKM_CAST3_CBC_PAD 0x00000315
2035 #define CKM_CAST5_KEY_GEN 0x00000320
2036 #define CKM_CAST128_KEY_GEN 0x00000320
2037 #define CKM_CAST5_ECB 0x00000321
2038 #define CKM_CAST128_ECB 0x00000321
2039 #define CKM_CAST5_CBC 0x00000322
2040 #define CKM_CAST128_CBC 0x00000322
2041 #define CKM_CAST5_MAC 0x00000323
2042 #define CKM_CAST128_MAC 0x00000323
2043 #define CKM_CAST5_MAC_GENERAL 0x00000324
2044 #define CKM_CAST128_MAC_GENERAL 0x00000324
2045 #define CKM_CAST5_CBC_PAD 0x00000325
2046 #define CKM_CAST128_CBC_PAD 0x00000325
2047 #define CKM_RC5_KEY_GEN 0x00000330
2048 #define CKM_RC5_ECB 0x00000331
2049 #define CKM_RC5_CBC 0x00000332
2050 #define CKM_RC5_MAC 0x00000333
2051 #define CKM_RC5_MAC_GENERAL 0x00000334
2052 #define CKM_RC5_CBC_PAD 0x00000335
2053 #define CKM_IDEA_KEY_GEN 0x00000340
2054 #define CKM_IDEA_ECB 0x00000341
2055 #define CKM_IDEA_CBC 0x00000342
2056 #define CKM_IDEA_MAC 0x00000343
2057 #define CKM_IDEA_MAC_GENERAL 0x00000344
2058 #define CKM_IDEA_CBC_PAD 0x00000345
2059 #define CKM_MD5_KEY_DERIVATION 0x00000390
2060 #define CKM_MD2_KEY_DERIVATION 0x00000391
2061 #define CKM_PBE_MD2_DES_CBC 0x000003A0
2062 #define CKM_PBE_MD5_DES_CBC 0x000003A1
2063 #define CKM_PBE_MD5_CAST_CBC 0x000003A2
2064 #define CKM_PBE_MD5_CAST3_CBC 0x000003A3
2065 #define CKM_PBE_MD5_CAST5_CBC 0x000003A4
2066 #define CKM_PBE_MD5_CAST128_CBC 0x000003A4
2067 #define CKM_PBE_SHA1_CAST5_CBC 0x000003A5
2068 #define CKM_PBE_SHA1_CAST128_CBC 0x000003A5
2069 #define CKM_PBE_SHA1_RC4_128 0x000003A6
2070 #define CKM_PBE_SHA1_RC4_40 0x000003A7
2071 #define CKM_PBE_SHA1_RC2_128_CBC 0x000003AA
2072 #define CKM_PBE_SHA1_RC2_40_CBC 0x000003AB
2073 #define CKM_KEY_WRAP_LYNKS 0x00000400
2074 #define CKM_KEY_WRAP_SET_OAEP 0x00000401
2075 #define CKM_SKIPJACK_KEY_GEN 0x00001000
2076 #define CKM_SKIPJACK_ECB64 0x00001001
2077 #define CKM_SKIPJACK_CBC64 0x00001002
2078 #define CKM_SKIPJACK_OFB64 0x00001003
2079 #define CKM_SKIPJACK_CFB64 0x00001004
2080 #define CKM_SKIPJACK_CFB32 0x00001005
2081 #define CKM_SKIPJACK_CFB16 0x00001006
2082 #define CKM_SKIPJACK_CFB8 0x00001007
2083 #define CKM_SKIPJACK_WRAP 0x00001008
2084 #define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009
2085 #define CKM_SKIPJACK_RELAYX 0x0000100a
2086 #define CKM_KEA_KEY_PAIR_GEN 0x00001010
2087 #define CKM_KEA_KEY_DERIVE 0x00001011
2088 #define CKM_FORTEZZA_TIMESTAMP 0x00001020
2089 #define CKM_BATON_KEY_GEN 0x00001030
2090 #define CKM_BATON_ECB128 0x00001031
2091 #define CKM_BATON_ECB96 0x00001032
2092 #define CKM_BATON_CBC128 0x00001033
2093 #define CKM_BATON_COUNTER 0x00001034

```

```
2094 #define CKM_BATON_SHUFFLE 0x00001035
2095 #define CKM_BATON_WRAP 0x00001036
2096 #define CKM_JUNIPER_KEY_GEN 0x00001060
2097 #define CKM_JUNIPER_ECB128 0x00001061
2098 #define CKM_JUNIPER_CBC128 0x00001062
2099 #define CKM_JUNIPER_COUNTER 0x00001063
2100 #define CKM_JUNIPER_SHUFFLE 0x00001064
2101 #define CKM_JUNIPER_WRAP 0x00001065
2102 #define CKM_FASTHASH 0x00001070
```

2103

2104

Appendix C. Revision History

2105

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

2106