



# PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40

Committee Specification ~~0102~~

16 ~~November~~ September 2014

## Specification URIs

### This version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs02/pkcs11-hist-v2.40-cs02.doc>  
(Authoritative)  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs02/pkcs11-hist-v2.40-cs02.html>  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs02/pkcs11-hist-v2.40-cs02.pdf>

### Previous version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.doc>  
(Authoritative)  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.html>  
<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs01/pkcs11-hist-v2.40-cs01.pdf>

### Previous version:

~~(Authoritative)~~

## 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](mailto:robert.griffin@rsa.com)), EMC Corporation  
Valerie Fenwick ([valerie.fenwick@oracle.com](mailto:valerie.fenwick@oracle.com)), Oracle

## Editors:

Susan Gleeson ([susan.gleeson@oracle.com](mailto:susan.gleeson@oracle.com)), Oracle  
Chris Zimman ([chris@wmpp.com](mailto:chris@wmpp.com)), Individual

## 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. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=pkcs11#technical](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=pkcs11#technical).

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC’s public comment list, after subscribing to it by following the instructions at the “[Send A Comment](#)” button on the TC’s web page at <https://www.oasis-open.org/committees/pkcs11/>.

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

**Citation format:**

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

**[PKCS11-Hist-v2.40]**

*PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40*. Edited by Susan Gleeson and Chris Zimman. 16 ~~November~~ <sup>September</sup> 2014. OASIS Committee Specification ~~0402~~. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/cs02/pkcs11-hist-v2.40-cs02.html>. Latest version: <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/pkcs11-hist-v2.40.html>.

---

## Notices

Copyright © OASIS Open 2014. All Rights Reserved.

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

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

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

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

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

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

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

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

---

# Table of Contents

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

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

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

2.18.1	Definitions .....	59
2.18.2	LYNKS key wrapping .....	59
3	PKCS #11 Implementation Conformance .....	60
Appendix A.	Acknowledgments .....	61
Appendix B.	Manifest constants .....	64
Appendix C.	Revision History .....	67

---

# 1 Introduction

## 1.1 Description of this Document

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

All text is normative unless otherwise labeled.

## 1.2 Terminology

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

## 1.3 Definitions

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

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



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

## 48 1.4 Normative References

49	<b>[PKCS #11-Base]</b>	<i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i> Edited by Susan Gleeson and Chris Zimman. <del>16 September 2014. OASIS Committee Specification 01.</del> Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html</a> .
53	<b>[PKCS #11-Curr]</b>	<i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40.</i> Edited by Susan Gleeson and Chris Zimman. <del>16 September 2014. OASIS Committee Specification 01.</del> Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html</a> .
59	<b>[PKCS #11-Prof]</b>	<i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i> Edited by Tim Hudson. <del>16 September 2014. OASIS Committee Specification 01.</del> Latest version: <a href="http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html</a> .
64	<b>[RFC2119]</b>	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> .
66		

## 67 1.5 Non-Normative References

68	<b>[ANSI C]</b>	ANSI/ISO. American National Standard for Programming Languages – C. 1990
69	<b>[ANSI X9.31]</b>	Accredited Standards Committee X9. Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA). 1998.
71	<b>[ANSI X9.42]</b>	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography. 2003
74	<b>[ANSI X9.62]</b>	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998
76	<b>[CC/PP]</b>	G. Klyne, F. Reynolds, C. , H. Ohto, J. Hjelm, M. H. Butler, L. Tran, Editors, W3C. <i>Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies</i> . 2004, URL: <a href="http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/">http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/</a>
80	<b>[CDPD]</b>	Ameritech Mobile Communications et al. Cellular Digital Packet Data System Specifications: Part 406: Airlink Security. 1993
82	<b>[FIPS PUB 46-3]</b>	NIST. <i>FIPS 46-3: Data Encryption Standard (DES)</i> . October 26, 2999. URL: <a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
83		

84 **[FIPS PUB 81]** NIST. *FIPS 81: DES Modes of Operation*. December 1980. URL:  
85 <http://csrc.nist.gov/publications/fips/index.html>

86 **[FIPS PUB 113]** NIST. *FIPS 113: Computer Data Authentication*. May 30, 1985. URL:  
87 <http://csrc.nist.gov/publications/fips/index.html>

88 **[FIPS PUB 180-2]** NIST. *FIPS 180-2: Secure Hash Standard*. August 1, 2002. URL:  
89 <http://csrc.nist.gov/publications/fips/index.html>

90 **[FORTEZZA CIPG]** NSA, Workstation Security Products. *FORTEZZA Cryptologic Interface*  
91 *Programmers Guide, Revision 1.52*. November 1985

92 **[GCS-API]** X/Open Company Ltd. *Generic Cryptographic Service API (GCS-API), Base –*  
93 *Draft 2*. February 14, 1995.

94 **[ISO/IEC 7816-1]** ISO/IEC 7816-1:2011. *Identification Cards – Integrated circuit cards -- Part 1:*  
95 *Cards with contacts -- Physical Characteristics*. 2011 URL:  
96 [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=54089](http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089).

97 **[ISO/IEC 7816-4]** ISO/IEC 7618-4:2013. *Identification Cards – Integrated circuit cards – Part 4:*  
98 *Organization, security and commands for interchange*. 2013. URL:  
99 [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=54550](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54550).

100

101 **[ISO/IEC 8824-1]** ISO/IEC 8824-1:2008. *Abstract Syntax Notation One (ASN.1): Specification of*  
102 *Base Notation*. 2002. URL:  
103 [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)  
104 [54012](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012)

105 **[ISO/IEC 8825-1]** ISO/IEC 8825-1:2008. *Information Technology – ASN.1 Encoding Rules:*  
106 *Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER),*  
107 *and Distinguished Encoding Rules (DER)*. 2008. URL:  
108 [http://www.iso.org/iso/home/store/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=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)  
109

110 **[ISO/IEC 9594-1]** ISO/IEC 9594-1:2008. *Information Technology – Open System Interconnection –*  
111 *The Directory: Overview of Concepts, Models and Services*. 2008. URL:  
112 [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=53364](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53364)  
113

114 **[ISO/IEC 9594-8]** ISO/IEC 9594-8:2008. *Information Technology – Open Systems Interconnection*  
115 *– The Directory: Public-key and Attribute Certificate Frameworks*. 2008 URL:  
116 [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=53372](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53372)  
117

118 **[ISO/IEC 9796-2]** ISO/IEC 9796-2:2010. *Information Technology – Security Techniques – Digital*  
119 *Signature Scheme Giving Message Recovery – Part 2: Integer factorization*  
120 *based mechanisms*. 2010. URL:  
121 [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=54788](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788)  
122

123 **[Java MIDP]** Java Community Process. *Mobile Information Device Profile for Java 2 Micro*  
124 *Edition*. November 2002. URL: <http://jcp.org/jsr/detail/118.jsp>

125 **[MeT-PTD]** MeT. *MeT PTD Definition – Personal Trusted Device Definition, Version 1.0*.  
126 February 2003. URL: <http://www.mobiletransaction.org>

127 **[PCMCIA]** Personal Computer Memory Card International Association. *PC Card Standard,*  
128 *Release 2.1*. July 1993.

129 **[PKCS #1]** RSA Laboratories. *RSA Cryptography Standard, v2.1*. June 14, 2002 URL:  
130 <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>

131 **[PKCS #3]** RSA Laboratories. *Diffie-Hellman Key-Agreement Standard, v1.4*. November  
132 1993.

133 **[PKCS #5]** RSA Laboratories. *Password-Based Encryption Standard, v2.0*. March 26,  
134 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf>

135 **[PKCS #7]** RSA Laboratories. *Cryptographic Message Syntax Standard, v1.6*. November  
136 1997 URL : <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v16.pdf>

137 **[PKCS #8]** RSA Laboratories. *Private-Key Information Syntax Standard, v1.2*. November  
138 1993. URL : [ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1\\_2.asn](ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn)

139 **[PKCS #11-UG]** *PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40*. Edited by  
140 John Leiseboer and Robert Griffin. ~~16 September 2014. OASIS Committee Note~~  
141 ~~04.~~ Latest version: <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.  
142

143 **[PKCS #12]** RSA Laboratories. *Personal Information Exchange Syntax Standard, v1.0*.  
144 June 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf>

145 **[RFC 1321]** R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. MIT Laboratory for  
146 Computer Science and RSA Data Security, Inc., April 1992. URL:  
147 <http://www.rfc-editor.org/rfc/rfc1321.txt>

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

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

152 **[SEC-1]** Standards for Efficient Cryptography Group (SECG). *Standards for Efficient*  
153 *Cryptography (SEC) 1: Elliptic Curve Cryptography*. Version 1.0, September 20,  
154 2000.

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

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

160 **[WIM]** WAP. *Wireless Identity Module. – WAP-260-WIP-20010712.a*. July 2001. URL:  
161 <http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf>  
162

163 **[WPKI]** WAP. *Wireless Application Protocol: Public Key Infrastructure Definition. – WAP-*  
164 *217-WPKI-20010424-a*. April 2001. URL:  
165 <http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf>  
166

167 **[WTLS]** WAP. *Wireless Transport Layer Security Version – WAP-261-WTLS-20010406-*  
168 *a*. April 2001. URL:  
169 <http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf>  
170

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

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

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

179 **[X.690]** ITU-T. *Information Technology – ASN.1 Encoding Rules: Specification of Basic*  
180 *Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished*  
181 *Encoding Rules (DER)*. July 2002. (Identical to ISO/IEC 8825-1)  
182

183

## 2 Mechanisms

184

### 2.1 PKCS #11 Mechanisms

185

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.

186

187

188

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

193

194

195

Table 1, Mechanisms vs. Functions

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

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	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 <sup>3</sup>	
CKM_BATON_KEY_GEN					X		
CKM_BATON_ECB128	X						
CKM_BATON_ECB96	X						

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

196 <sup>1</sup> SR = SignRecover, VR = VerifyRecover.

197 <sup>2</sup> Single-part operations only.

198 <sup>3</sup> Mechanism MUST only be used for wrapping, not unwrapping.

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

201 In general, if a mechanism makes no mention of the *ulMinKeyLen* and *ulMaxKeyLen* fields of the  
202 CK\_MECHANISM\_INFO structure, then those fields have no meaning for that particular mechanism.  
203

## 204 2.2 FORTEZZA timestamp

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

208 **It has no parameters.**

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

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

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

212 <sup>1</sup> Single-part operations only

213 <sup>2</sup> Data length, signature length

214 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
215 specify the supported range of DSA prime sizes, in bits.

## 216 2.3 KEA

### 217 2.3.1 Definitions

218 This section defines the key type “CKK\_KEA” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE  
219 attribute of key objects.

220 Mechanisms:

221 CKM\_KEA\_KEY\_PAIR\_GEN

222 CKM\_KEA\_KEY\_DERIVE

### 223 2.3.2 KEA mechanism parameters

#### 224 2.3.2.1 CK\_KEA\_DERIVE\_PARAMS; CK\_KEA\_DERIVE\_PARAMS\_PTR

225

226 **CK\_KEA\_DERIVE\_PARAMS** is a structure that provides the parameters to the **CKM\_KEA\_DERIVE**  
227 mechanism. It is defined as follows:

```
228 typedef struct CK_KEA_DERIVE_PARAMS {  
229     CK_BBOOL isSender;  
230     CK_ULONG ulRandomLen;  
231     CK_BYTE_PTR pRandomA;  
232     CK_BYTE_PTR pRandomB;  
233     CK_ULONG ulPublicDataLen;  
234     CK_BYTE_PTR pPublicData;  
235 } CK_KEA_DERIVE_PARAMS;
```

236

237 The fields of the structure have the following meanings:

238 *isSender* Option for generating the key (called a TEK). The value  
 239 is CK\_TRUE if the sender (originator) generates the  
 240 TEK, CK\_FALSE if the recipient is regenerating the TEK

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

242 *pRandomA* pointer to Ra data

243 *pRandomB* pointer to Rb data

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

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

246 **CK\_KEA\_DERIVE\_PARAMS\_PTR** is a pointer to a **CK\_KEA\_DERIVE\_PARAMS**.

### 247 2.3.3 KEA public key objects

248 KEA public key objects (object class **CKO\_PUBLIC\_KEY**, key type **CKK\_KEA**) hold KEA public keys.  
 249 The following table defines the KEA public key object attributes, in addition to the common attributes  
 250 defined for this object class:

251 *Table 3, KEA Public Key Object Attributes*

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

252 Refer to [PKCS #11-Base] table [4510](#) for footnotes

253 The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the "KEA domain  
 254 parameters".

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

```

256 CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
257 CK_KEY_TYPE keyType = CKK_KEA;
258 CK_UTF8CHAR label[] = "A KEA public key object";
259 CK_BYTE prime[] = {...};
260 CK_BYTE subprime[] = {...};
261 CK_BYTE base[] = {...};
262 CK_BYTE value[] = {...};
263 CK_ATTRIBUTE template[] = {
264     {CKA_CLASS, &class, sizeof(class)},
265     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
266     {CKA_TOKEN, &>true, sizeof(true)},
267     {CKA_LABEL, label, sizeof(label)-1},
268     {CKA_PRIME, prime, sizeof(prime)},
269     {CKA_SUBPRIME, subprime, sizeof(subprime)},
270     {CKA_BASE, base, sizeof(base)},
271     {CKA_VALUE, value, sizeof(value)}
272 };
  
```

273



## 274 2.3.4 KEA private key objects

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

278 *Table 4, KEA Private Key Object Attributes*

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

279 Refer to [PKCS #11-Base] table [4510](#) for footnotes

280

281 The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the “KEA domain  
282 parameters”.

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

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

```
287 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;  
288 CK_KEY_TYPE keyType = CKK_KEA;  
289 CK_UTF8CHAR label[] = "A KEA private key object";  
290 CK_BYTE subject[] = {...};  
291 CK_BYTE id[] = {123};  
292 CK_BYTE prime[] = {...};  
293 CK_BYTE subprime[] = {...};  
294 CK_BYTE base[] = {...};  
295 CK_BYTE value[] = {...};  
296 CK_BBOOL true = CK_TRUE;  
297 CK_ATTRIBUTE template[] = {  
298     {CKA_CLASS, &class, sizeof(class)},  
299     {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS  
300     {CKA_TOKEN, &>true, sizeof(true)},  
301     {CKA_LABEL, label, sizeof(label) - 1},  
302     {CKA_SUBJECT, subject, sizeof(subject)},  
303     {CKA_ID, id, sizeof(id)},  
304     {CKA_SENSITIVE, &true, sizeof(true)},  
305     {CKA_DERIVE, &true, sizeof(true)},  
306     {CKA_PRIME, prime, sizeof(prime)},  
307     {CKA_SUBPRIME, subprime, sizeof(subprime)},  
308     {CKA_BASE, base, sizeof(base)},  
309     {CKA_VALUE, value, sizeof(value)}  
310 };
```

## 311 2.3.5 KEA key pair generation

312 The KEA key pair generation mechanism, denoted **CKM\_KEA\_KEY\_PAIR\_GEN**, generates key pairs for  
313 the Key Exchange Algorithm, as defined by NIST’s “SKIPJACK and KEA Algorithm Specification Version  
314 2.0”, 29 May 1998.

315 It does not have a parameter.

316 The mechanism generates KEA public/private key pairs with a particular prime, subprime and base, as  
317 specified in the **CKA\_PRIME**, **CKA\_SUBPRIME**, and **CKA\_BASE** attributes of the template for the public

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

320 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE** and **CKA\_VALUE** attributes to the new  
 321 public key and the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_SUBPRIME**, **CKA\_BASE**, and  
 322 **CKA\_VALUE** attributes to the new private key. Other attributes supported by the KEA public and private  
 323 key types (specifically, the flags indicating which functions the keys support) MAY also be specified in the  
 324 templates for the keys, or else are assigned default initial values.

325 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 326 specify the supported range of KEA prime sizes, in bits.

327 **2.3.6 KEA key derivation**

328 The KEA key derivation mechanism, denoted **CKM\_DEA\_DERIVE**, is a mechanism for key derivation  
 329 based on KEA, the Key Exchange Algorithm, as defined by NIST's "SKIPJACK and KEA Algorithm  
 330 Specification Version 2.0", 29 May 1998.

331 It has a parameter, a **CK\_KEA\_DERIVE\_PARAMS** structure.

332 This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE**  
 333 attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of  
 334 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism  
 335 contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key  
 336 type must be specified in the template.

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

342 The operation of this mechanism depends on two of the values in the supplied  
 343 **CK\_KEA\_DERIVE\_PARAMS** structure, as detailed in the table below. Note that in all cases, the data  
 344 buffers pointed to by the parameter structure fields *pRandomA* and *pRandomB* must be allocated by the  
 345 caller prior to invoking **C\_DeriveKey**. Also, the values pointed to by *pRandomA* and *pRandomB* are  
 346 represented as Cryptoki "Big integer" data (i.e., a sequence of bytes, most significant byte first).

347 *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.

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

350 key. In these cases, any object template supplied as the **C\_DeriveKey** *pTemplate* argument should be  
351 ignored.

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

- 353 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY  
354 both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on  
355 some default value.
- 356 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived  
357 key MUST as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to  
358 CK\_TRUE, then the derived has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value  
359 as its **CKA\_SENSITIVE** attribute.
- 360 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then  
361 the derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set  
362 to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the  
363 *opposite* value from its **CKA\_EXTRACTABLE** attribute.

364 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
365 specify the supported range of KEA prime sizes, in bits.

## 366 2.4 RC2

### 367 2.4.1 Definitions

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

373 This section defines the key type “CKK\_RC2” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE  
374 attribute of key objects.

375 Mechanisms:

- 376 CKM\_RC2\_KEY\_GEN
- 377 CKM\_RC2\_ECB
- 378 CKM\_RC2\_CBC
- 379 CKM\_RC2\_MAC
- 380 CKM\_RC2\_MAC\_GENERAL
- 381 CKM\_RC2\_CBC\_PAD

### 382 2.4.2 RC2 secret key objects

383 RC2 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC2**) hold RC2 keys. The  
384 following table defines the RC2 secret key object attributes, in addition to the common attributes defined  
385 for this object class:

386 *Table 6, RC2 Secret Key Object Attributes*

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

\* Note that the rules regarding the **CKA\_SENSITIVE**, **CKA\_EXTRACTABLE**,  
**CKA\_ALWAYS\_SENSITIVE**, and **CKA\_NEVER\_EXTRACTABLE** attributes have changed in version  
2.11 to match the policy used by other key derivation mechanisms such as  
**CKM\_SSL3\_MASTER\_KEY\_DERIVE**.

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

387 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

```

389 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
390 CK_KEY_TYPE keyType = CKK_RC2;
391 CK_UTF8CHAR label[] = "An RC2 secret key object";
392 CK_BYTE value[] = {...};
393 CK_BBOOL true = CK_TRUE;
394 CK_ATTRIBUTE template[] = {
395     {CKA_CLASS, &class, sizeof(class)},
396     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
397     {CKA_TOKEN, &>true, sizeof(true)},
398     {CKA_LABEL, label, sizeof(label)-1},
399     {CKA_ENCRYPT, &>true, sizeof(true)},
400     {CKA_VALUE, value, sizeof(value)}
401 };

```

## 402 2.4.3 RC2 mechanism parameters

### 403 2.4.3.1 CK\_RC2\_PARAMS; CK\_RC2\_PARAMS\_PTR

404 **CK\_RC2\_PARAMS** provides the parameters to the **CKM\_RC2\_ECB** and **CKM\_RC2\_MAC** mechanisms.  
405 It holds the effective number of bits in the RC2 search space. It is defined as follows:

```

406 typedef CK_ULONG CK_RC2_PARAMS;

```

407 **CK\_RC2\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_PARAMS**.

### 408 2.4.3.2 CK\_RC2\_CBC\_PARAMS; CK\_RC2\_CBC\_PARAMS\_PTR

409 **CK\_RC2\_CBC\_PARAMS** is a structure that provides the parameters to the **CKM\_RC2\_CBC** and  
410 **CKM\_RC2\_CBC\_PAD** mechanisms. It is defined as follows:

```

411 typedef struct CK_RC2_CBC_PARAMS {
412     CK_ULONG ulEffectiveBits;
413     CK_BYTE iv[8];
414 } CK_RC2_CBC_PARAMS;

```

415 The fields of the structure have the following meanings:

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

417 *iv* the initialization vector (IV) for cipher block chaining

418 mode

419 **CK\_RC2\_CBC\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_CBC\_PARAMS**.

### 420 2.4.3.3 CK\_RC2\_MAC\_GENERAL\_PARAMS; 421 CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR

422 **CK\_RC2\_MAC\_GENERAL\_PARAMS** is a structure that provides the parameters to the  
423 **CKM\_RC2\_MAC\_GENERAL** mechanism. It is defined as follows:

```

424 typedef struct CK_RC2_MAC_GENERAL_PARAMS {
425     CK_ULONG ulEffectiveBits;
426     CK_ULONG ulMacLength;
427 } CK_RC2_MAC_GENERAL_PARAMS;

```

428 The fields of the structure have the following meanings:  
 429 *ulEffectiveBits* the effective number of bits in the RC2 search space  
 430 *ulMacLength* length of the MAC produced, in bytes  
 431 **CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_MAC\_GENERAL\_PARAMS**.

#### 432 2.4.4 RC2 key generation

433 The RC2 key generation mechanism, denoted **CKM\_RC2\_KEY\_GEN**, is a key generation mechanism for  
 434 RSA Security's block cipher RC2.

435 It does not have a parameter.

436 The mechanism generates RC2 keys with a particular length in bytes, as specified in the  
 437 **CKA\_VALUE\_LEN** attribute of the template for the key.

438 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 439 key. Other attributes supported by the RC2 key type (specifically, the flags indicating which functions the  
 440 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

441 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 442 specify the supported range of RC2 key sizes, in bits.

#### 443 2.4.5 RC2-ECB

444 RC2-ECB, denoted **CKM\_RC2\_ECB**, is a mechanism for single- and multiple-part encryption and  
 445 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and electronic  
 446 codebook mode as defined in FIPS PUB 81.

447 It has a parameter, a **CK\_RC2\_PARAMS**, which indicates the effective number of bits in the RC2 search  
 448 space.

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

455 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
 456 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
 457 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
 458 attribute of the new key; other attributes required by the key type must be specified in the template.

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

460 *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	

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

## 463 2.4.6 RC2-CBC

464 RC2\_CBC, denoted **CKM\_RC2\_CBC**, is a mechanism for single- and multiple-part encryption and  
465 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC2 and cipher-  
466 block chaining mode as defined in FIPS PUB 81.

467 It has a parameter, a **CK\_RC2\_CBC\_PARAMS** structure, where the first field indicates the effective  
468 number of bits in the RC2 search space, and the next field is the initialization vector for cipher block  
469 chaining mode.

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

476 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
477 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
478 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
479 attribute of the new key; other attributes required by the key type must be specified in the template.

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

481 *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	

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

## 484 2.4.7 RC2-CBC with PKCS padding

485 RC2-CBC with PKCS padding, denoted **CKM\_RC2\_CBC\_PAD**, is a mechanism for single- and multiple-  
486 part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher  
487 RC2; cipher-block chaining mode as defined in FIPS PUB 81; and the block cipher padding method  
488 detailed in PKCS #7.

489 It has a parameter, a **CK\_RC2\_CBC\_PARAMS** structure, where the first field indicates the effective  
490 number of bits in the RC2 search space, and the next field is the initialization vector.

491 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
492 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
493 for the **CKA\_VALUE\_LEN** attribute.

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

497 for data length constraints when wrapping and unwrapping keys do not apply to wrapping and  
498 unwrapping private keys.

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

500 *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

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

### 503 **2.4.8 General-length RC2-MAC**

504 General-length RC2-MAC, denoted **CKM\_RC2\_MAC\_GENERAL**, is a mechanism for single-and  
505 multiple-part signatures and verification, based on RSA Security's block cipher RC2 and data  
506 authorization as defined in FIPS PUB 113.

507 It has a parameter, a **CK\_RC2\_MAC\_GENERAL\_PARAMS** structure, which specifies the effective  
508 number of bits in the RC2 search space and the output length desired from the mechanism.

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

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

512 *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

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 **2.4.9 RC2-MAC**

516 RC2-MAC, denoted by **CKM\_RC2\_MAC**, is a special case of the general-length RC2-MA mechanism  
517 (see Section 2.4.8). Instead of taking a **CK\_RC2\_MAC\_GENERAL\_PARAMS** parameter, it takes a  
518 **CK\_RC2\_PARAMS** parameter, which only contains the effective number of bits in the RC2 search space.  
519 RC2-MAC produces and verifies 4-byte MACs.

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

521

522 *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

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

## 525 2.5 RC4

### 526 2.5.1 Definitions

527 This section defines the key type “CKK\_RC4” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE  
528 attribute of key objects.

529 Mechanisms

530 CKM\_RC4\_KEY\_GEN

531 CKM\_RC4

### 532 2.5.2 RC4 secret key objects

533 RC4 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC4**) hold RC4 keys. The  
534 following table defines the RC4 secret key object attributes, in addition to the common attributes defined  
535 for this object class:

536 *Table 12, RC4 Secret Key Object*

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

537 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

```
539 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
540 CK_KEY_TYPE keyType = CKK_RC4;  
541 CK_UTF8CHAR label[] = "An RC4 secret key object";  
542 CK_BYTE value[] = {...};  
543 CK_BBOOL true = CK_TRUE;  
544 CK_ATTRIBUTE template[] = {  
545     {CKA_CLASS, &class, sizeof(class)},  
546     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
547     {CKA_TOKEN, &>true, sizeof(true)},  
548     {CKA_LABEL, label, sizeof(label)-1},  
549     {CKA_ENCRYPT, &>true, sizeof(true)},  
550     {CKA_VALUE, value, sizeof(value)}  
551 };
```

### 552 2.5.3 RC4 key generation

553 The RC4 key generation mechanism, denoted **CKM\_RC4\_KEY\_GEN**, is a key generation mechanism for  
554 RSA Security's proprietary stream cipher RC4.

555 It does not have a parameter.

556 The mechanism generates RC4 keys with a particular length in bytes, as specified in the  
557 **CKA\_VALUE\_LEN** attribute of the template for the key.

558 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
559 key. Other attributes supported by the RC4 key type (specifically, the flags indicating which functions the  
560 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

561 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
562 specify the supported range of RC4 key sizes, in bits.



## 563 2.5.4 RC4 mechanism

564 RC4, denoted **CKM\_RC4**, is a mechanism for single- and multiple-part encryption and decryption based  
565 on RSA Security's proprietary stream cipher RC4.

566 It does not have a parameter.

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

568 *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

569 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
570 specify the supported range of RC4 key sizes, in bits.

## 571 2.6 RC5

### 572 2.6.1 Definitions

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

575 This section defines the key type "CKK\_RC5" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE  
576 attribute of key objects.

577 Mechanisms:

578 CKM\_RC5\_KEY\_GEN

579 CKM\_RC5\_ECB

580 CKM\_RC5\_CBC

581 CKM\_RC5\_MAC

582 CKM\_RC5\_MAC\_GENERAL

583 CMK\_RC5\_CBC\_PAD

### 584 2.6.2 RC5 secret key objects

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

588 *Table 14, RC5 Secret Key Object*

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

589 Refer to [PKCS #11-Base] table [4510](#) for footnotes

590

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

```
592 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
593 CK_KEY_TYPE keyType = CKK_RC5;  
594 CK_UTF8CHAR label[] = "An RC5 secret key object";  
595 CK_BYTE value[] = {...};  
596 CK_BBOOL true = CK_TRUE;
```

```

597 CK_ATTRIBUTE template[] = {
598     {CKA_CLASS, &class, sizeof(class)},
599     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
600     {CKA_TOKEN, &>true, sizeof(true)},
601     {CKA_LABEL, label, sizeof(label)-1},
602     {CKA_ENCRYPT, &>true, sizeof(true)},
603     {CKA_VALUE, value, sizeof(value)}
604 };

```

## 605 2.6.3 RC5 mechanism parameters

### 606 2.6.3.1 CK\_RC5\_PARAMS; CK\_RC5\_PARAMS\_PTR

607 **CK\_RC5\_PARAMS** provides the parameters to the **CKM\_RC5\_ECB** and **CKM\_RC5\_MAC** mechanisms.  
608 It is defined as follows:

```

609 typedef struct CK_RC5_PARAMS {
610     CK_ULONG ulWordsize;
611     CK_ULONG ulRounds;
612 } CK_RC5_PARAMS;

```

613 The fields of the structure have the following meanings:

614 *ulWordsize*      wordsize of RC5 cipher in bytes

615 *ulRounds*        number of rounds of RC5 encipherment

616 **CK\_RC5\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_PARAMS**.

### 617 2.6.3.2 CK\_RC5\_CBC\_PARAMS; CK\_RC5\_CBC\_PARAMS\_PTR

618 **CK\_RC5\_CBC\_PARAMS** is a structure that provides the parameters to the **CKM\_RC5\_CBC** and  
619 **CKM\_RC5\_CBC\_PAD** mechanisms. It is defined as follows:

```

620 typedef struct CK_RC5_CBC_PARAMS {
621     CK_ULONG ulWordsize;
622     CK_ULONG ulRounds;
623     CK_BYTE_PTR pIv;
624     CK_ULONG ulIvLen;
625 } CK_RC5_CBC_PARAMS;

```

626 The fields of the structure have the following meanings:

627 *ulwordSize*      wordsize of RC5 cipher in bytes

628 *ulRounds*        number of rounds of RC5 encipherment

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

630 *ulIvLen*         length of initialization vector (must be same as  
631                    blocksize)

632 **CK\_RC5\_CBC\_PARAMS\_PTR** is a pointer to a **CK\_RC5\_CBC\_PARAMS**.

### 633 2.6.3.3 CK\_RC5\_MAC\_GENERAL\_PARAMS; 634 CK\_RC5\_MAC\_GENERAL\_PARAMS\_PTR

635 **CK\_RC5\_MAC\_GENERAL\_PARAMS** is a structure that provides the parameters to the  
636 **CKM\_RC5\_MAC\_GENERAL** mechanism. It is defined as follows:

```

637 typedef struct CK_RC5_MAC_GENERAL_PARAMS {
638     CK_ULONG ulWordsize;
639     CK_ULONG ulRounds;
640     CK_ULONG ulMacLength;
641 } CK_RC5_MAC_GENERAL_PARAMS;

```

642 The fields of the structure have the following meanings:

- 643 *ulwordSize*      wordsize of RC5 cipher in bytes
- 644 *ulRounds*        number of rounds of RC5 encipherment
- 645 *ulMacLength*     length of the MAC produced, in bytes

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

## 647 2.6.4 RC5 key generation

648 The RC5 key generation mechanism, denoted **CKM\_RC5\_KEY\_GEN**, is a key generation mechanism for  
649 RSA Security's block cipher RC5.

650 It does not have a parameter.

651 The mechanism generates RC5 keys with a particular length in bytes, as specified in the  
652 **CKA\_VALUE\_LEN** attribute of the template for the key.

653 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
654 key. Other attributes supported by the RC5 key type (specifically, the flags indicating which functions the  
655 key supports) MAY be specified in the template for the key, or else are assigned default initial values.

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

## 658 2.6.5 RC5-ECB

659 RC5-ECB, denoted **CKM\_RC5\_ECB**, is a mechanism for single- and multiple-part encryption and  
660 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and electronic  
661 codebook mode as defined in FIPS PUB 81.

662 It has a parameter, **CK\_RC5\_PARAMS**, which indicates the wordsize and number of rounds of  
663 encryption to use.

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

670 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
671 **CKA\_KEY\_TYPE** attributes of the template and, if it has one, and the key type supports it, the  
672 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
673 attribute of the new key; other attributes required by the key type must be specified in the template.

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

675 *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	

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

## 678 2.6.6 RC5-CBC

679 RC5-CBC, denoted **CKM\_RC5\_CBC**, is a mechanism for single- and multiple-part encryption and  
680 decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher RC5 and cipher-  
681 block chaining mode as defined in FIPS PUB 81.

682 It has a parameter, a **CK\_RC5\_CBC\_PARAMS** structure, which specifies the wordsize and number of  
683 rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.

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

690 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
691 **CKA\_KEY\_TYPE** attribute for the template, and, if it has one, and the key type supports it, the  
692 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
693 attribute of the new key; other attributes required by the key type must be specified in the template.

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

695 *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	

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

## 698 2.6.7 RC5-CBC with PKCS padding

699 RC5-CBC with PKCS padding, denoted **CKM\_RC5\_CBC\_PAD**, is a mechanism for single- and multiple-  
700 part encryption and decryption; key wrapping; and key unwrapping, based on RSA Security's block cipher  
701 RC5; cipher block chaining mode as defined in FIPS PUB 81; and the block cipher padding method  
702 detailed in PKCS #7.

703 It has a parameter, a **CK\_RC5\_CBC\_PARAMS** structure, which specifies the wordsize and number of  
 704 rounds of encryption to use, as well as the initialization vector for cipher block chaining mode.

705 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
 706 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
 707 for the **CKA\_VALUE\_LEN** attribute.

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

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

713 *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

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

## 716 2.6.8 General-length RC5-MAC

717 General-length RC5-MAC, denoted **CKM\_RC5\_MAC\_GENERAL**, is a mechanism for single- and  
 718 multiple-part signatures and verification, based on RSA Security's block cipher RC5 and data  
 719 authentication as defined in FIPS PUB 113.

720 It has a parameter, a **CK\_RC5\_MAC\_GENERAL\_PARAMS** structure, which specifies the wordsize and  
 721 number of rounds of encryption to use and the output length desired from the mechanism.

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

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

725 *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

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

## 728 2.6.9 RC5-MAC

729 RC5-MAC, denoted by **CKM\_RC5\_MAC**, is a special case of the general-length RC5-MAC mechanism.  
 730 Instead of taking a **CK\_RC5\_MAC\_GENERAL\_PARAMS** parameter, it takes a **CK\_RC5\_PARAMS**  
 731 parameter. RC5-MAC produces and verifies MACs half as large as the RC5 blocksize.

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

733 *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]

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

## 736 2.7 General block cipher

### 737 2.7.1 Definitions

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

741 This section defines the key types "CKK\_DES", "CKK\_CAST", "CKK\_CAST3", "CKK\_CAST5"  
742 (deprecated in v2.11), "CKK\_CAST128", "CKK\_IDEA" and "CKK\_CDMF" for type CK\_KEY\_TYPE as  
743 used in the CKA\_KEY\_TYPE attribute of key objects.

744 Mechanisms:

745 CKM\_DES\_KEY\_GEN  
746 CKM\_DES\_ECB  
747 CKM\_DES\_CBC  
748 CKM\_DES\_MAC  
749 CKM\_DES\_MAC\_GENERAL  
750 CKM\_DES\_CBC\_PAD  
751 CKM\_CDMF\_KEY\_GEN  
752 CKM\_CDMF\_ECB  
753 CKM\_CDMF\_CBC  
754 CKM\_CDMF\_MAC  
755 CKM\_CDMF\_MAC\_GENERAL  
756 CKM\_CDMF\_CBC\_PAD  
757 CKM\_DES\_OFB64  
758 CKM\_DES\_OFB8  
759 CKM\_DES\_CFB64  
760 CKM\_DES\_CFB8  
761 CKM\_CAST\_KEY\_GEN  
762 CKM\_CAST\_ECB  
763 CKM\_CAST\_CBC  
764 CKM\_CAST\_MAC  
765 CKM\_CAST\_MAC\_GENERAL  
766 CKM\_CAST\_CBC\_PAD  
767 CKM\_CAST3\_KEY\_GEN  
768 CKM\_CAST3\_ECB  
769 CKM\_CAST3\_CBC  
770 CKM\_CAST3\_MAC  
771 CKM\_CAST3\_MAC\_GENERAL

772 CKM\_CAST3\_CBC\_PAD  
 773 CKM\_CAST5\_KEY\_GEN  
 774 CKM\_CAST128\_KEY\_GEN  
 775 CKM\_CAST5\_ECB  
 776 CKM\_CAST128\_ECB  
 777 CKM\_CAST5\_CBC  
 778 CKM\_CAST128\_CBC  
 779 CKM\_CAST5\_MAC  
 780 CKM\_CAST128\_MAC  
 781 CKM\_CAST5\_MAC\_GENERAL  
 782 CKM\_CAST128\_MAC\_GENERAL  
 783 CKM\_CAST5\_CBC\_PAD  
 784 CKM\_CAST128\_CBC\_PAD  
 785 CKM\_IDEA\_KEY\_GEN  
 786 CKM\_IDEA\_ECB  
 787 CKM\_IDEA\_MAC  
 788 CKM\_IDEA\_MAC\_GENERAL  
 789 CKM\_IDEA\_CBC\_PAD

790 **2.7.2 DES secret key objects**

791 DES secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_DES**) hold single-length DES  
 792 keys. The following table defines the DES secret key object attributes, in addition to the common  
 793 attributes defined for this object class:

794 *Table 20, DES Secret Key Object*

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

795 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

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

```

799 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
800 CK_KEY_TYPE keyType = CKK_DES;
801 CK_UTF8CHAR label[] = "A DES secret key object";
802 CK_BYTE value[8] = {...};
803 CK_BBOOL true = CK_TRUE;
804 CK_ATTRIBUTE template[] = {
805     {CKA_CLASS, &class, sizeof(class)},
806     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
807     {CKA_TOKEN, &>true, sizeof(true)},
808     {CKA_LABEL, label, sizeof(label)-1},
809     {CKA_ENCRYPT, &>true, sizeof(true)},
810     {CKA_VALUE, value, sizeof(value)}
811 };
  
```

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

## 815 2.7.3 CAST secret key objects

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

819 *Table 21, CAST Secret Key Object Attributes*

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

820 Refer to [PKCS #11-Base] table [4510](#) for footnotes

821

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

```
823 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
824 CK_KEY_TYPE keyType = CKK_CAST;  
825 CK_UTF8CHAR label[] = "A CAST secret key object";  
826 CK_BYTE value[] = {...};  
827 CK_BBOOL true = CK_TRUE;  
828 CK_ATTRIBUTE template[] = {  
829     {CKA_CLASS, &class, sizeof(class)},  
830     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
831     {CKA_TOKEN, &>true, sizeof(true)},  
832     {CKA_LABEL, label, sizeof(label)-1},  
833     {CKA_ENCRYPT, &>true, sizeof(true)},  
834     {CKA_VALUE, value, sizeof(value)}  
835 };
```

## 836 2.7.4 CAST3 secret key objects

837 CAST3 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_CAST3**) hold CAST3 keys.  
838 The following table defines the CAST3 secret key object attributes, in addition to the common attributes  
839 defines for this object class:

840 *Table 22, CAST3 Secret Key Object Attributes*

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

841 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

```
843 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
844 CK_KEY_TYPE keyType = CKK_CAST3;  
845 CK_UTF8CHAR label[] = "A CAST3 secret key object";  
846 CK_BYTE value[] = {...};  
847 CK_BBOOL true = CK_TRUE;  
848 CK_ATTRIBUTE template[] = {  
849     {CKA_CLASS, &class, sizeof(class)},  
850     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
851     {CKA_TOKEN, &>true, sizeof(true)},  
852     {CKA_LABEL, label, sizeof(label)-1},  
853     {CKA_ENCRYPT, &>true, sizeof(true)},  
854     {CKA_VALUE, value, sizeof(value)}  
855 };
```



## 856 2.7.5 CAST128 (CAST5) secret key objects

857 CAST128 (also known as CAST5) secret key objects (object class **CKO\_SECRET\_KEY**, key type  
858 **CKK\_CAST128** or **CKK\_CAST5**) hold CAST128 keys. The following table defines the CAST128 secret  
859 key object attributes, in addition to the common attributes defines for this object class:

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

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

861 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

```
863 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
864 CK_KEY_TYPE keyType = CKK_CAST128;  
865 CK_UTF8CHAR label[] = "A CAST128 secret key object";  
866 CK_BYTE value[] = {...};  
867 CK_BBOOL true = CK_TRUE;  
868 CK_ATTRIBUTE template[] = {  
869     {CKA_CLASS, &class, sizeof(class)},  
870     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
871     {CKA_TOKEN, &true, sizeof(true)},  
872     {CKA_LABEL, label, sizeof(label)-1},  
873     {CKA_ENCRYPT, &true, sizeof(true)},  
874     {CKA_VALUE, value, sizeof(value)}  
875 };
```

876

## 877 2.7.6 IDEA secret key objects

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

880 Table 24, IDEA Secret Key Object

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

881 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

```
883 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
884 CK_KEY_TYPE keyType = CKK_IDEA;  
885 CK_UTF8CHAR label[] = "An IDEA secret key object";  
886 CK_BYTE value[16] = {...};  
887 CK_BBOOL true = CK_TRUE;  
888 CK_ATTRIBUTE template[] = {  
889     {CKA_CLASS, &class, sizeof(class)},  
890     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
891     {CKA_TOKEN, &true, sizeof(true)},  
892     {CKA_LABEL, label, sizeof(label)-1},  
893     {CKA_ENCRYPT, &true, sizeof(true)},  
894     {CKA_VALUE, value, sizeof(value)}  
895 };
```

896

## 897 2.7.7 CDMF secret key objects

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

900 *Table 25, CDMF Secret Key Object*

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

901 Refer to [PKCS #11-Base] table [4510](#) for footnotes

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

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

```
906 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
907 CK_KEY_TYPE keyType = CKK_CDMF;  
908 CK_UTF8CHAR label[] = "A CDMF secret key object";  
909 CK_BYTE value[8] = {...};  
910 CK_BBOOL true = CK_TRUE;  
911 CK_ATTRIBUTE template[] = {  
912     {CKA_CLASS, &class, sizeof(class)},  
913     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
914     {CKA_TOKEN, &>true, sizeof(true)},  
915     {CKA_LABEL, label, sizeof(label)-1},  
916     {CKA_ENCRYPT, &>true, sizeof(true)},  
917     {CKA_VALUE, value, sizeof(value)}  
918 };
```

## 919 2.7.8 General block cipher mechanism parameters

### 920 2.7.8.1 CK\_MAC\_GENERAL\_PARAMS; CK\_MAC\_GENERAL\_PARAMS\_PTR

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

```
927 typedef CK_ULONG CK_MAC_GENERAL_PARAMS;  
928
```

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

## 930 2.7.9 General block cipher key generation

931 Cipher <NAME> has a key generation mechanism, "<NAME> key generation", denoted by  
932 **CKM\_<NAME>\_KEY\_GEN**.

933 This mechanism does not have a parameter.

934 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
935 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key  
936 supports) MAY be specified in the template for the key, or else are assigned default initial values.

937 When DES keys or CDMF keys are generated, their parity bits are set properly, as specified in FIPS PUB  
938 46-3. Similarly, when a triple-DES key is generated, each of the DES keys comprising it has its parity bits  
939 set properly.

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

943 When CAST, CAST3, or CAST128 (CAST5) keys are generated, the template for the secret key must  
 944 specify a **CKA\_VALUE\_LEN** attribute.

945 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 946 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for  
 947 the key generation mechanisms for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the  
 948 **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES,  
 949 DES3 (triple-DES), IDEA and CDMF ciphers, these fields and not used.

## 950 2.7.10 General block cipher ECB

951 Cipher <NAME> has an electronic codebook mechanism, “<NAME>-ECB”, denoted  
 952 **CKM\_<NAME>\_ECB**. It is a mechanism for single- and multiple-part encryption and decryption; key  
 953 wrapping; and key unwrapping with <NAME>.

954 It does not have a parameter.

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

961 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
 962 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
 963 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
 964 attribute of the new key; other attributes required by the key must be specified in the template.

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

966 *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	

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

## 972 2.7.11 General block cipher CBC

973 Cipher <NAME> has a cipher-block chaining mode, “<NAME>-CBC”, denoted **CKM\_<NAME>\_CBC**. It is  
 974 a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping  
 975 with <NAME>.

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

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

979 *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	

980 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 981 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for  
 982 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 983 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA, and CDMF  
 984 ciphers, these fields are not used.

## 985 2.7.12 General block cipher CBC with PKCS padding

986 Cipher <NAME> has a cipher-block chaining mode with PKCS padding, “<NAME>-CBC with PKCS  
 987 padding”, denoted **CKM\_<NAME>\_CBC\_PAD**. It is a mechanism for single- and multiple-part encryption  
 988 and decryption; key wrapping; and key unwrapping with <NAME>. All ciphertext is padded with PKCS  
 989 padding.

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

992 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
 993 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
 994 for the **CKA\_VALUE\_LEN** attribute.

995

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

1000 Constraints on key types and the length of data are summarized in the following table:

1001 *Table 28, General Block Cipher CBC with PKCS Padding: Key and Data Length*

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

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

1002 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
1003 MAY be used. The CAST, CAST3 and CAST128 (CAST5) ciphers have variable key sizes, and so for  
1004 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
1005 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA, and CDMF  
1006 ciphers, these fields are not used.

### 1007 2.7.13 General-length general block cipher MAC

1008 Cipher <NAME> has a general-length MACing mode, “General-length <NAME>-MAC”, denoted  
1009 **CKM\_<NAME>\_MAC\_GENERAL**. It is a mechanism for single-and multiple-part signatures and  
1010 verification, based on the <NAME> encryption algorithm and data authentication as defined in FIPS PUB  
1011 113.

1012 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which specifies the size of the output.

1013 The output bytes from this mechanism are taken from the start of the final cipher block produced in the  
1014 MACing process.

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

1016 *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

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

### 1022 2.7.14 General block cipher MAC

1023 Cipher <NAME> has a MACing mechanism, “<NAME>-MAC”, denoted **CKM\_<NAME>\_MAC**. This  
1024 mechanism is a special case of the **CKM\_<NAME>\_MAC\_GENERAL** mechanism described above. It  
1025 produces an output of size half as large as <NAME>’s blocksize.

1026 This mechanism has no parameters.

1027 Constraints on key types and the length of data are summarized in the following table:

1028 *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]

1029 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
1030 MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for  
1031 these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
1032 specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF  
1033 ciphers, these fields are not used.

## 1034 2.8 SKIPJACK

### 1035 2.8.1 Definitions

1036 This section defines the key type “CKK\_SKIPJACK” for type CK\_KEY\_TYPE as used in the  
1037 CKA\_KEY\_TYPE attribute of key objects.

1038 Mechanisms:

- 1039 CKM\_SKIPJACK\_KEY\_GEN
- 1040 CKM\_SKIPJACK\_ECB64
- 1041 CKM\_SKIPJACK\_CBC64
- 1042 CKM\_SKIPJACK\_OFB64
- 1043 CKM\_SKIPJACK\_CFB64
- 1044 CKM\_SKIPJACK\_CFB32
- 1045 CKM\_SKIPJACK\_CFB16
- 1046 CKM\_SKIPJACK\_CFB8
- 1047 CKM\_SKIPJACK\_WRAP
- 1048 CKM\_SKIPJACK\_PRIVATE\_WRAP
- 1049 CKM\_SKIPJACK\_RELAYX

### 1050 2.8.2 SKIPJACK secret key objects

1051 SKIPJACK secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_SKIPJACK**) holds a  
1052 single-length MEK or a TEK. The following table defines the SKIPJACK secret object attributes, in  
1053 addition to the common attributes defined for this object class:

1054 *Table 31, SKIPJACK Secret Key Object*

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

1055 Refer to [PKCS #11-Base] table [4510](#) for footnotes

1056

1057 SKIPJACK keys have 16 checksum bits, and these bits must be properly set. Attempting to create or  
1058 unwrap a SKIPJACK key with incorrect checksum bits MUST return an error.

1059 It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK  
1060 key with a specified value. Nonetheless, we provide templates for doing so.

1061 The following is a sample template for creating a SKIPJACK MEK secret key object:

```
1062 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1063 CK_KEY_TYPE keyType = CKK_SKIPJACK;  
1064 CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";  
1065 CK_BYTE value[12] = {...};  
1066 CK_BBOOL true = CK_TRUE;  
1067 CK_ATTRIBUTE template[] = {  
1068     {CKA_CLASS, &class, sizeof(class)},  
1069     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1070     {CKA_TOKEN, &>true, sizeof(true)},  
1071     {CKA_LABEL, label, sizeof(label)-1},  
1072     {CKA_ENCRYPT, &>true, sizeof(true)},  
1073     {CKA_VALUE, value, sizeof(value)}  
1074 };
```

1075 The following is a sample template for creating a SKIPJACK TEK secret key object:

```

1076 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1077 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1078 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1079 CK_BYTE value[12] = {...};
1080 CK_BBOOL true = CK_TRUE;
1081 CK_ATTRIBUTE template[] = {
1082     {CKA_CLASS, &class, sizeof(class)},
1083     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1084     {CKA_TOKEN, &true, sizeof(true)},
1085     {CKA_LABEL, label, sizeof(label)-1},
1086     {CKA_ENCRYPT, &true, sizeof(true)},
1087     {CKA_WRAP, &true, sizeof(true)},
1088     {CKA_VALUE, value, sizeof(value)}
1089 };

```

## 1090 2.8.3 SKIPJACK Mechanism parameters

### 1091 2.8.3.1 CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS; 1092 CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR

1093 **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS** is a structure that provides the parameters to the  
1094 **CKM\_SKIPJACK\_PRIVATE\_WRAP** mechanism. It is defined as follows:

```

1095 typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
1096     CK_ULONG ulPasswordLen;
1097     CK_BYTE_PTR pPassword;
1098     CK_ULONG ulPublicDataLen;
1099     CK_BYTE_PTR pPublicData;
1100     CK_ULONG ulPandGLen;
1101     CK_ULONG ulQLen;
1102     CK_ULONG ulRandomLen;
1103     CK_BYTE_PTR pRandomA;
1104     CK_BYTE_PTR pPrimeP;
1105     CK_BYTE_PTR pBaseG;
1106     CK_BYTE_PTR pSubprimeQ;
1107 } CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

```

1108 The fields of the structure have the following meanings:

1109	<i>ulPasswordLen</i>	length of the password
1110	<i>pPassword</i>	pointer to the buffer which contains the user-supplied
1111		password
1112	<i>ulPublicDataLen</i>	other party's key exchange public key size
1113	<i>pPublicData</i>	pointer to other party's key exchange public key value
1114	<i>ulPandGLen</i>	length of prime and base values
1115	<i>ulQLen</i>	length of subprime value
1116	<i>ulRandomLen</i>	size of random Ra, in bytes
1117	<i>pPrimeP</i>	pointer to Prime, p, value
1118	<i>pBaseG</i>	pointer to Base, b, value

1119 *pSubprimeQ* pointer to Subprime, q, value

1120 **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR** is a pointer to a  
1121 **CK\_PRIVATE\_WRAP\_PARAMS**.

### 1122 **2.8.3.2 CK\_SKIPJACK\_RELAYX\_PARAMS;** 1123 **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR**

1124 **CK\_SKIPJACK\_RELAYX\_PARAMS** is a structure that provides the parameters to the  
1125 **CKM\_SKIPJACK\_RELAYX** mechanism. It is defined as follows:

```
1126 typedef struct CK_SKIPJACK_RELAYX_PARAMS {  
1127     CK_ULONG ulOldWrappedXLen;  
1128     CK_BYTE_PTR pOldWrappedX;  
1129     CK_ULONG ulOldPasswordLen;  
1130     CK_BYTE_PTR pOldPassword;  
1131     CK_ULONG ulOldPublicDataLen;  
1132     CK_BYTE_PTR pOldPublicData;  
1133     CK_ULONG ulOldRandomLen;  
1134     CK_BYTE_PTR pOldRandomA;  
1135     CK_ULONG ulNewPasswordLen;  
1136     CK_BYTE_PTR pNewPassword;  
1137     CK_ULONG ulNewPublicDataLen;  
1138     CK_BYTE_PTR pNewPublicData;  
1139     CK_ULONG ulNewRandomLen;  
1140     CK_BYTE_PTR pNewRandomA;  
1141 } CK_SKIPJACK_RELAYX_PARAMS;
```

1142 The fields of the structure have the following meanings:

1143 *ulOldWrappedLen* length of old wrapped key in bytes

1144 *pOldWrappedX* pointer to old wrapper key

1145 *ulOldPasswordLen* length of the old password

1146 *pOldPassword* pointer to the buffer which contains the old user-supplied  
1147 password

1148 *ulOldPublicDataLen* old key exchange public key size

1149 *pOldPublicData* pointer to old key exchange public key value

1150 *ulOldRandomLen* size of old random Ra in bytes

1151 *pOldRandomA* pointer to old Ra data

1152 *ulNewPasswordLen* length of the new password

1153 *pNewPassword* pointer to the buffer which contains the new user-  
1154 supplied password

1155 *ulNewPublicDataLen* new key exchange public key size

1156 *pNewPublicData* pointer to new key exchange public key value



1157 *ulNewRandomLen* size of new random Ra in bytes

1158 *pNewRandomA* pointer to new Ra data

1159 **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR** is a pointer to a **CK\_SKIPJACK\_RELAYX\_PARAMS**.

## 1160 2.8.4 SKIPJACK key generation

1161 The SKIPJACK key generation mechanism, denoted **CKM\_SKIPJACK\_KEY\_GEN**, is a key generation  
1162 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1163 It does not have a parameter.

1164 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1165 key.

## 1166 2.8.5 SKIPJACK-ECB64

1167 SKIPJACK-ECB64, denoted **CKM\_SKIPJACK\_ECB64**, is a mechanism for single- and multiple-part  
1168 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB  
1169 185.

1170 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1171 value generated by the token – in other words, the application cant specify a particular IV when  
1172 encrypting. It MAY, of course, specify a particular IV when decrypting.

1173 Constraints on key types and the length of data are summarized in the following table:

1174 *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

## 1175 2.8.6 SKIPJACK-CBC64

1176 SKIPJACK-CBC64, denoted **CKM\_SKIPJACK\_CBC64**, is a mechanism for single- and multiple-part  
1177 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1178 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1179 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1180 encrypting. It MAY, of course, specify a particular IV when decrypting.

1181 Constraints on key types and the length of data are summarized in the following table:

1182 *Table 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

## 1183 2.8.7 SKIPJACK-OFB64

1184 SKIPJACK-OFB64, denoted **CKM\_SKIPJACK\_OFB64**, is a mechanism for single- and multiple-part  
1185 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

1186 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1187 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1188 encrypting. It MAY, of course, specify a particular IV when decrypting.

1189 Constraints on key types and the length of data are summarized in the following table:

1190 *Table 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

### 1191 2.8.8 SKIPJACK-CFB64

1192 SKIPJACK-CFB64, denoted **CKM\_SKIPJACK\_CFB64**, is a mechanism for single- and multiple-part  
1193 encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.

1194 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1195 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1196 encrypting. It MAY, of course, specify a particular IV when decrypting.

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

1198 *Table 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

### 1199 2.8.9 SKIPJACK-CFB32

1200 SKIPJACK-CFB32, denoted **CKM\_SKIPJACK\_CFB32**, is a mechanism for single- and multiple-part  
1201 encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.

1202 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1203 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1204 encrypting. It MAY, of course, specify a particular IV when decrypting.

1205 Constraints on key types and the length of data are summarized in the following table:

1206 *Table 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

### 1207 2.8.10 SKIPJACK-CFB16

1208 SKIPJACK-CFB16, denoted **CKM\_SKIPJACK\_CFB16**, is a mechanism for single- and multiple-part  
1209 encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.

1210 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1211 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1212 encrypting. It MAY, of course, specify a particular IV when decrypting.

1213 Constraints on key types and the length of data are summarized in the following table:

1214 *Table 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
-----------	----------	---------------	----------------------	---------------

## 1215 2.8.11 SKIPJACK-CFB8

1216 SKIPJACK-CFB8, denoted **CKM\_SKIPJACK\_CFB8**, is a mechanism for single- and multiple-part  
1217 encryption and decryption with SKIPJACK in 8-bit cipher feedback mode as defined in FIPS PUB 185.

1218 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1219 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1220 encrypting. It MAY, of course, specify a particular IV when decrypting.

1221 Constraints on key types and the length of data are summarized in the following table:

1222 *Table 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

## 1223 2.8.12 SKIPJACK-WRAP

1224 The SKIPJACK-WRAP mechanism, denoted **CKM\_SKIPJACK\_WRAP**, is used to wrap and unwrap a  
1225 secret key (MEK). It MAY wrap or unwrap SKIPJACK, BATON, and JUNIPER keys.

1226 It does not have a parameter.

## 1227 2.8.13 SKIPJACK-PRIVATE-WRAP

1228 The SKIPJACK-PRIVATE-WRAP mechanism, denoted **CKM\_SKIPJACK\_PRIVATE\_WRAP**, is used to  
1229 wrap and unwrap a private key. It MAY wrap KEA and DSA private keys.

1230 It has a parameter, a **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS** structure.

## 1231 2.8.14 SKIPJACK-RELAYX

1232 The SKIPJACK-RELAYX mechanism, denoted **CKM\_SKIPJACK\_RELAYX**, is used with the **C\_WrapKey**  
1233 function to “change the wrapping” on a private key which was wrapped with the SKIPJACK-PRIVATE-  
1234 WRAP mechanism (See Section 2.8.13).

1235 It has a parameter, a **CK\_SKIPJACK\_RELAYX\_PARAMS** structure.

1236 Although the SKIPJACK-RELAYX mechanism is used with **C\_WrapKey**, it differs from other key-  
1237 wrapping mechanisms. Other key-wrapping mechanisms take a key handle as one of the arguments to  
1238 **C\_WrapKey**; however for the SKIPJACK\_RELAYX mechanism, the [always invalid] value 0 should be  
1239 passed as the key handle for **C\_WrapKey**, and the already-wrapped key should be passed in as part of  
1240 the **CK\_SKIPJACK\_RELAYX\_PARAMS** structure.

## 1241 2.9 BATON

### 1242 2.9.1 Definitions

1243 This section defines the key type “CKK\_BATON” for type CK\_KEY\_TYPE as used in the  
1244 CKA\_KEY\_TYPE attribute of key objects.

1245 Mechanisms:

1246 CKM\_BATON\_KEY\_GEN

1247 CKM\_BATON\_ECB128

1248 CKM\_BATON\_ECB96

1249 CKM\_BATON\_CBC128  
1250 CKM\_BATON\_COUNTER  
1251 CKM\_BATON\_SHUFFLE  
1252 CKM\_BATON\_WRAP

## 1253 2.9.2 BATON secret key objects

1254 BATON secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_BATON**) hold single-length  
1255 BATON keys. The following table defines the BATON secret key object attributes, in addition to the  
1256 common attributes defined for this object class:

1257 *Table 39, BATON Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (40 bytes long)

1258 Refer to [PKCS #11-Base] table [4510](#) for footnotes

1259

1260 BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or  
1261 unwrap a BATON key with incorrect checksum bits MUST return an error.

1262 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key  
1263 with a specified value. Nonetheless, we provide templates for doing so.

1264 The following is a sample template for creating a BATON MEK secret key object:

```
1265 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1266 CK_KEY_TYPE keyType = CKK_BATON;  
1267 CK_UTF8CHAR label[] = "A BATON MEK secret key object";  
1268 CK_BYTE value[40] = {...};  
1269 CK_BBOOL true = CK_TRUE;  
1270 CK_ATTRIBUTE template[] = {  
1271     {CKA_CLASS, &class, sizeof(class)},  
1272     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1273     {CKA_TOKEN, &>true, sizeof(true)},  
1274     {CKA_LABEL, label, sizeof(label)-1},  
1275     {CKA_ENCRYPT, &>true, sizeof(true)},  
1276     {CKA_VALUE, value, sizeof(value)}  
1277 };
```

1278 The following is a sample template for creating a BATON TEK secret key object:

```
1279 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1280 CK_KEY_TYPE keyType = CKK_BATON;  
1281 CK_UTF8CHAR label[] = "A BATON TEK secret key object";  
1282 CK_BYTE value[40] = {...};  
1283 CK_BBOOL true = CK_TRUE;  
1284 CK_ATTRIBUTE template[] = {  
1285     {CKA_CLASS, &class, sizeof(class)},  
1286     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1287     {CKA_TOKEN, &>true, sizeof(true)},  
1288     {CKA_LABEL, label, sizeof(label)-1},  
1289     {CKA_ENCRYPT, &>true, sizeof(true)},  
1290     {CKA_WRAP, &>true, sizeof(true)},  
1291     {CKA_VALUE, value, sizeof(value)}  
1292 };
```

## 1293 2.9.3 BATON key generation

1294 The BATON key generation mechanism, denoted **CKM\_BATON\_KEY\_GEN**, is a key generation  
1295 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

1296 It does not have a parameter.  
 1297 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 1298 key.

## 1299 **2.9.4 BATON-ECB128**

1300 BATON-ECB128, denoted **CKM\_BATON\_ECB128**, is a mechanism for single- and multiple-part  
 1301 encryption and decryption with BATON in 128-bit electronic codebook mode.  
 1302 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
 1303 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
 1304 encrypting. It MAY, of course, specify a particular IV when decrypting.  
 1305 Constraints on key types and the length of data are summarized in the following table:

1306 *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

## 1307 **2.9.5 BATON-ECB96**

1308 BATON-ECB96, denoted **CKM\_BATON\_ECB96**, is a mechanism for single- and multiple-part encryption  
 1309 and decryption with BATON in 96-bit electronic codebook mode.  
 1310 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
 1311 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
 1312 encrypting. It MAY, of course, specify a particular IV when decrypting.

1313 Constraints on key types and the length of data are summarized in the following table:

1314 *Table 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

## 1315 **2.9.6 BATON-CBC128**

1316 BATON-CBC128, denoted **CKM\_BATON\_CBC128**, is a mechanism for single- and multiple-part  
 1317 encryption and decryption with BATON in 128-bit cipher-block chaining mode.  
 1318 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
 1319 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
 1320 encrypting. It MAY, of course, specify a particular IV when decrypting.

1321 Constraints on key types and the length of data are summarized in the following table:

1322 *Table 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

## 1323 2.9.7 BATON-COUNTER

1324 BATON-COUNTER, denoted **CKM\_BATON\_COUNTER**, is a mechanism for single- and multiple-part  
1325 encryption and decryption with BATON in counter mode.

1326 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1327 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1328 encrypting. It MAY, of course, specify a particular IV when decrypting.

1329 Constraints on key types and the length of data are summarized in the following table:

1330 *Table 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

## 1331 2.9.8 BATON-SHUFFLE

1332 BATON-SHUFFLE, denoted **CKM\_BATON\_SHUFFLE**, is a mechanism for single- and multiple-part  
1333 encryption and decryption with BATON in shuffle mode.

1334 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1335 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1336 encrypting. It MAY, of course, specify a particular IV when decrypting.

1337 Constraints on key types and the length of data are summarized in the following table:

1338 *Table 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

## 1339 2.9.9 BATON WRAP

1340 The BATON wrap and unwrap mechanism, denoted **CKM\_BATON\_WRAP**, is a function used to wrap  
1341 and unwrap a secret key (MEK). It MAY wrap and unwrap SKIPJACK, BATON and JUNIPER keys.

1342 It has no parameters.

1343 When used to unwrap a key, this mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and  
1344 **CKA\_VALUE** attributes to it.

## 1345 2.10 JUNIPER

### 1346 2.10.1 Definitions

1347 This section defines the key type “CKK\_JUNIPER” for type CK\_KEY\_TYPE as used in the  
1348 CKA\_KEY\_TYPE attribute of key objects.

1349 Mechanisms:

1350 CKM\_JUNIPER\_KEY\_GEN

1351 CKM\_JUNIPER\_ECB128

1352 CKM\_JUNIPER\_CBC128

1353 CKM\_JUNIPER\_COUNTER

1354 CKM\_JUNIPER\_SHUFFLE

1355 CKM\_JUNIPER\_WRAP

## 1356 2.10.2 JUNIPER secret key objects

1357 JUNIPER secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_JUNIPER**) hold single-  
1358 length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to  
1359 the common attributes defined for this object class:

1360 *Table 45, JUNIPER Secret Key Object*

Attribute	Data type	Meaning
CKA_VALUE <sup>1,4,6,7</sup>	Byte array	Key value (40 bytes long)

1361 Refer to [PKCS #11-Base] table [4510](#) for footnotes

1362

1363 JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or  
1364 unwrap a BATON key with incorrect checksum bits MUST return an error.

1365 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key  
1366 with a specified value. Nonetheless, we provide templates for doing so.

1367 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1368 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1369 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1370 CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";  
1371 CK_BYTE value[40] = {...};  
1372 CK_BBOOL true = CK_TRUE;  
1373 CK_ATTRIBUTE template[] = {  
1374     {CKA_CLASS, &class, sizeof(class)},  
1375     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1376     {CKA_TOKEN, &>true, sizeof(true)},  
1377     {CKA_LABEL, label, sizeof(label)-1},  
1378     {CKA_ENCRYPT, &>true, sizeof(true)},  
1379     {CKA_VALUE, value, sizeof(value)}  
1380 };
```

1381 The following is a sample template for creating a JUNIPER TEK secret key object:

```
1382 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1383 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1384 CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";  
1385 CK_BYTE value[40] = {...};  
1386 CK_BBOOL true = CK_TRUE;  
1387 CK_ATTRIBUTE template[] = {  
1388     {CKA_CLASS, &class, sizeof(class)},  
1389     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1390     {CKA_TOKEN, &>true, sizeof(true)},  
1391     {CKA_LABEL, label, sizeof(label)-1},  
1392     {CKA_ENCRYPT, &>true, sizeof(true)},  
1393     {CKA_WRAP, &>true, sizeof(true)},  
1394     {CKA_VALUE, value, sizeof(value)}  
1395 };
```

## 1396 2.10.3 JUNIPER key generation

1397 The JUNIPER key generation mechanism, denoted **CKM\_JUNIPER\_KEY\_GEN**, is a key generation  
1398 mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).

1399 It does not have a parameter.

1400 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1401 key.

## 1402 2.10.4 JUNIPER-ECB128

1403 JUNIPER-ECB128, denoted **CKM\_JUNIPER\_ECB128**, is a mechanism for single- and multiple-part  
1404 encryption and decryption with JUNIPER in 128-bit electronic codebook mode.

1405 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1406 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1407 encrypting. It MAY, of course, specify a particular IV when decrypting.

1408 Constraints on key types and the length of data are summarized in the following table. For encryption  
1409 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1410 *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

## 1411 2.10.5 JUNIPER-CBC128

1412 JUNIPER-CBC128, denoted **CKM\_JUNIPER\_CBC128**, is a mechanism for single- and multiple-part  
1413 encryption and decryption with JUNIPER in 128-bit cipher block chaining mode.

1414 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1415 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1416 encrypting. It MAY, of course, specify a particular IV when decrypting.

1417 Constraints on key types and the length of data are summarized in the following table. For encryption  
1418 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1419 *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

## 1420 2.10.6 JUNIPER-COUNTER

1421 JUNIPER-COUNTER, denoted **CKM\_JUNIPER\_COUNTER**, is a mechanism for single- and multiple-  
1422 part encryption and decryption with JUNIPER in counter mode.

1423 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1424 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1425 encrypting. It MAY, of course, specify a particular IV when decrypting.

1426 Constraints on key types and the length of data are summarized in the following table. For encryption  
1427 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1428 *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

## 1429 2.10.7 JUNIPER-SHUFFLE

1430 JUNIPER-SHUFFLE, denoted **CKM\_JUNIPER\_SHUFFLE**, is a mechanism for single- and multiple-part  
1431 encryption and decryption with JUNIPER in shuffle mode.



1432 It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some  
1433 value generated by the token – in other words, the application MAY NOT specify a particular IV when  
1434 encrypting. It MAY, of course, specify a particular IV when decrypting.

1435 Constraints on key types and the length of data are summarized in the following table. For encryption  
1436 and decryption, the input and output data (parts) MAY begin at the same location in memory.

1437 *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

## 1438 2.10.8 JUNIPER WRAP

1439 The JUNIPER wrap and unwrap mechanism, denoted **CKM\_JUNIPER\_WRAP**, is a function used to wrap  
1440 and unwrap an MEK. It MAY wrap or unwrap SKIPJACK, BATON and JUNIPER keys.

1441 It has no parameters.

1442 When used to unwrap a key, this mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and  
1443 **CKA\_VALUE** attributes to it.

## 1444 2.11 MD2

### 1445 2.11.1 Definitions

1446 Mechanisms:

1447 CKM\_MD2

1448 CKM\_MD2\_HMAC

1449 CKM\_MD2\_HMAC\_GENERAL

1450 CKM\_MD2\_KEY\_DERIVATION

### 1451 2.11.2 MD2 digest

1452 The MD2 mechanism, denoted **CKM\_MD2**, is a mechanism for message digesting, following the MD2  
1453 message-digest algorithm defined in RFC 6149.

1454 It does not have a parameter.

1455 Constraints on the length of data are summarized in the following table:

1456 *Table 50, MD2: Data Length*

Function	Data length	Digest Length
C_Digest	Any	16

### 1457 2.11.3 General-length MD2-HMAC

1458 The general-length MD2-HMAC mechanism, denoted **CKM\_MD2\_HMAC\_GENERAL**, is a mechanism for  
1459 signatures and verification. It uses the HMAC construction, based on the MD2 hash function. The keys it  
1460 uses are generic secret keys.

1461 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1462 output. This length should be in the range 0-16 (the output size of MD2 is 16 bytes). Signatures (MACs)  
1463 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1464 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

## 1465 2.11.4 MD2-HMAC

1466 The MD2-HMAC mechanism, denoted **CKM\_MD2\_HMAC**, is a special case of the general-length MD2-  
1467 HMAC mechanism in Section 2.11.3.

1468 It has no parameter, and produces an output of length 16.

## 1469 2.11.5 MD2 key derivation

1470 MD2 key derivation, denoted **CKM\_MD2\_KEY\_DERIVATION**, is a mechanism which provides the  
1471 capability of deriving a secret key by digesting the value of another secret key with MD2.

1472 The value of the base key is digested once, and the result is used to make the value of the derived secret  
1473 key.

- 1474 • If no length or key type is provided in the template, then the key produced by this mechanism **MUST**  
1475 be a generic secret key. Its length **MUST** be 16 bytes (the output size of MD2)..
- 1476 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
1477 **MUST** be a generic secret key of the specified length.
- 1478 • If no length was provided in the template, but a key type is, then that key type must have a well-  
1479 defined length. If it does, then the key produced by this mechanism **MUST** be of the type specified in  
1480 the template. If it doesn't, an error **MUST** be returned.
- 1481 • If both a key type and a length are provided in the template, the length must be compatible with that  
1482 key type. The key produced by this mechanism **MUST** be of the specified type and length.

1483 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key **MUST** be set  
1484 properly.

1485 If the requested type of key requires more than 16 bytes, such as DES2, an error is generated.

1486 This mechanism has the following rules about key sensitivity and extractability:

- 1487 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key **MAY**  
1488 both be specified to be either **CK\_TRUE** or **CK\_FALSE**. If omitted, these attributes each take on  
1489 some default value.
- 1490 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to **CK\_FALSE**, then the derived key  
1491 **MUST** as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to **CK\_TRUE**, then  
1492 the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
1493 **CKA\_SENSITIVE** attribute.
- 1494 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_FALSE**, then the  
1495 derived key **MUST**, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
1496 **CK\_TRUE**, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
1497 value from its **CKA\_EXTRACTABLE** attribute.

## 1498 2.12 MD5

### 1499 2.12.1 Definitions

1500 Mechanisms:

1501 CKM\_MD5

1502 CKM\_MD5\_HMAC

1503 CKM\_MD5\_HMAC\_GENERAL  
1504 CKM\_MD5\_KEY\_DERIVATION

## 1505 2.12.2 MD5 Digest

1506 The MD5 mechanism, denoted **CKM\_MD5**, is a mechanism for message digesting, following the MD5  
1507 message-digest algorithm defined in RFC 1321.

1508 It does not have a parameter.

1509 Constraints on the length of input and output data are summarized in the following table. For single-part  
1510 digesting, the data and the digest MAY begin at the same location in memory.

1511 *Table 52, MD5: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

## 1512 2.12.3 General-length MD5-HMAC

1513 The general-length MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC\_GENERAL**, is a mechanism for  
1514 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it  
1515 uses are generic secret keys.

1516 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1517 output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)  
1518 produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

1519 *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

## 1520 2.12.4 MD5-HMAC

1521 The MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC**, is a special case of the general-length MD5-  
1522 HMAC mechanism in Section 2.12.3.

1523 It has no parameter, and produces an output of length 16.

## 1524 2.12.5 MD5 key derivation

1525 MD5 key derivation denoted **CKM\_MD5\_KEY\_DERIVATION**, is a mechanism which provides the  
1526 capability of deriving a secret key by digesting the value of another secret key with MD5.

1527 The value of the base key is digested once, and the result is used to make the value of derived secret  
1528 key.

- 1529
- 1530 • If no length or key type is provided in the template, then the key produced by this mechanism MUST be a generic secret key. Its length MUST be 16 bytes (the output size of MD5).
  - 1531 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
1532 MUST be a generic secret key of the specified length.
  - 1533 • If no length was provided in the template, but a key type is, then that key type must have a well-  
1534 defined length. If it does, then the key produced by this mechanism MUST be of the type specified in  
1535 the template. If it doesn't, an error MUST be returned.
  - 1536 • If both a key type and a length are provided in the template, the length must be compatible with that  
1537 key type. The key produced by this mechanism MUST be of the specified type and length.

1538 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST be set  
 1539 properly.

1540 If the requested type of key requires more than 16 bytes, such as DES3, an error is generated.

1541 This mechanism has the following rules about key sensitivity and extractability.

- 1542 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key MAY  
 1543 both be specified to either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
 1544 default value.
- 1545 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
 1546 MUST as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then  
 1547 the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
 1548 **CKA\_SENSITIVE** attribute.
- 1549 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
 1550 derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
 1551 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
 1552 value from its **CKA\_EXTRACTABLE** attribute.

## 1553 2.13 FASTHASH

### 1554 2.13.1 Definitions

1555 Mechanisms:  
 1556 CKM\_FASTHASH

### 1557 2.13.2 FASTHASH digest

1558 The FASTHASH mechanism, denoted **CKM\_FASTHASH**, is a mechanism for message digesting,  
 1559 following the U.S. government's algorithm.

1560 It does not have a parameter.

1561 Constraints on the length of input and output data are summarized in the following table:

1562 *Table 54, FASTHASH: Data Length*

Function	Input length	Digest length
C_Digest	Any	40

## 1563 2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD)

### 1564 2.14.1 Definitions

1565 The mechanisms in this section are for generating keys and IVs for performing password-based  
 1566 encryption. The method used to generate keys and IVs is specified in PKCS #5.

1567 Mechanisms:

- 1568 CKM\_PBE\_MD2\_DES\_CBC
- 1569 CKM\_PBE\_MD5\_DES\_CBC
- 1570 CKM\_PBE\_MD5\_CAST\_CBC
- 1571 CKM\_PBE\_MD5\_CAST3\_CBC
- 1572 CKM\_PBE\_MD5\_CAST5\_CBC
- 1573 CKM\_PBE\_MD5\_CAST128\_CBC
- 1574 CKM\_PBE\_SHA1\_CAST5\_CBC
- 1575 CKM\_PBE\_SHA1\_CAST128\_CBC

1576 CKM\_PBE\_SHA1\_RC4\_128  
1577 CKM\_PBE\_SHA1\_RC4\_40  
1578 CKM\_PBE\_SHA1\_RC2\_128\_CBC  
1579 CKM\_PBE\_SHA1\_RC2\_40\_CBC

## 1580 2.14.2 Password-based encryption/authentication mechanism parameters

### 1581 2.14.2.1 CK\_PBE\_PARAMS; CK\_PBE\_PARAMS\_PTR

1582 **CK\_PBE\_PARAMS** is a structure which provides all of the necessary information required by the  
1583 CKM\_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation  
1584 mechanisms) and the CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC mechanism. It is defined as follows:

```
1585 typedef struct CK_PBE_PARAMS {  
1586     CK_BYTE_PTR pInitVector;  
1587     CK_UTF8CHAR_PTR pPassword;  
1588     CK_ULONG ulPasswordLen;  
1589     CK_BYTE_PTR pSalt;  
1590     CK_ULONG ulSaltLen;  
1591     CK_ULONG ulIteration;  
1592 } CK_PBE_PARAMS;
```

1593 The fields of the structure have the following meanings:

1594	<i>pInitVector</i>	pointer to the location that receives the 8-byte initialization vector (IV), if an IV is required
1595		
1596	<i>pPassword</i>	points to the password to be used in the PBE key generation
1597		
1598	<i>ulPasswordLen</i>	length in bytes of the password information
1599	<i>pSalt</i>	points to the salt to be used in the PBE key generation
1600	<i>ulSaltLen</i>	length in bytes of the salt information
1601	<i>ulliteration</i>	number of iterations required for the generation

1602 **CK\_PBE\_PARAMS\_PTR** is a pointer to a **CK\_PBE\_PARAMS**.

### 1603 2.14.3 MD2-PBE for DES-CBC

1604 MD2-PBE for DES-CBC, denoted **CKM\_PBE\_MD2\_DES\_CBC**, is a mechanism used for generating a  
1605 DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an  
1606 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1607 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1608 key generation process and the location of the application-supplied buffer which receives the 8-byte  
1609 IV generated by the mechanism.

### 1610 2.14.4 MD5-PBE for DES-CBC

1611 MD5-PBE for DES-CBC, denoted **CKM\_PBE\_MD5\_DES\_CBC**, is a mechanism used for generating a  
1612 DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an  
1613 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1614 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1615 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1616 generated by the mechanism.

### 1617 **2.14.5 MD5-PBE for CAST-CBC**

1618 MD5-PBE for CAST-CBC, denoted **CKM\_PBE\_MD5\_CAST\_CBC**, is a mechanism used for generating a  
1619 CAST secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an  
1620 iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1621 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1622 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1623 generated by the mechanism

1624 The length of the CAST key generated by this mechanism MAY be specified in the supplied template; if it  
1625 is not present in the template, it defaults to 8 bytes.

### 1626 **2.14.6 MD5-PBE for CAST3-CBC**

1627 MD5-PBE for CAST3-CBC, denoted **CKM\_PBE\_MD5\_CAST3\_CBC**, is a mechanism used for generating  
1628 a CAST3 secret key and an IV from a password and a salt value by using the MD5 digest algorithm and  
1629 an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1630 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1631 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1632 generated by the mechanism

1633 The length of the CAST3 key generated by this mechanism MAY be specified in the supplied template; if  
1634 it is not present in the template, it defaults to 8 bytes.

### 1635 **2.14.7 MD5-PBE for CAST128-CBC (CAST5-CBC)**

1636 MD5-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM\_PBE\_MD5\_CAST128\_CBC** or  
1637 **CKM\_PBE\_MD5\_CAST5\_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key  
1638 and an IV from a password and a salt value by using the MD5 digest algorithm and an iteration count.  
1639 This functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1640 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1641 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1642 generated by the mechanism

1643 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied  
1644 template; if it is not present in the template, it defaults to 8 bytes.

### 1645 **2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)**

1646 SHA-1-PBE for CAST128-CBC (CAST5-CBC), denoted **CKM\_PBE\_SHA1\_CAST128\_CBC** or  
1647 **CKM\_PBE\_SHA1\_CAST5\_CBC**, is a mechanism used for generating a CAST128 (CAST5) secret key  
1648 and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This  
1649 functionality is analogous to that defined in PKCS #5 PBKDF1 for MD5 and DES.

1650 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1651 key generation process and the location of the application-supplied buffer which receives the 8-byte IV  
1652 generated by the mechanism

1653 The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied  
1654 template; if it is not present in the template, it defaults to 8 bytes

1655 **2.15 PKCS #12 password-based encryption/authentication**  
1656 **mechanisms**

1657 **2.15.1 Definitions**

1658 The mechanisms in this section are for generating keys and IVs for performing password-based  
1659 encryption or authentication. The method used to generate keys and IVs is based on a method that was  
1660 specified in PKCS #12.

1661 We specify here a general method for producing various types of pseudo-random bits from a password,  
1662  $p$ ; a string of salt bits,  $s$ ; and an iteration count,  $c$ . The “type” of pseudo-random bits to be produced is  
1663 identified by an identification byte,  $ID$ , described at the end of this section.

1664 Let  $H$  be a hash function built around a compression function  $f: Z_2^u \times Z_2^v \rightarrow Z_2^u$  (that is,  $H$  has a chaining  
1665 variable and output of length  $u$  bits, and the message input to the compression function of  $H$  is  $v$  bits). For  
1666 MD2 and MD5,  $u=128$  and  $v=512$ ; for SHA-1,  $u=160$  and  $v=512$ .

1667 We assume here that  $u$  and  $v$  are both multiples of 8, as are the lengths in bits of the password and salt  
1668 strings and the number  $n$  of pseudo-random bits required. In addition,  $u$  and  $v$  are of course nonzero.

- 1669 1. Construct a string,  $D$  (the “diversifier”), by concatenating  $v/8$  copies of  $ID$ .
- 1670 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  
1671 the salt MAY be truncated to create  $S$ ). Note that if the salt is the empty string, then so is  $S$ .
- 1672 3. Concatenate copies of the password together to create a string  $P$  of length  $v \cdot \lceil p/v \rceil$  bits (the final  
1673 copy of the password MAY be truncated to create  $P$ ). Note that if the password is the empty  
1674 string, then so is  $P$ .
- 1675 4. Set  $I=S||P$  to be the concatenation of  $S$  and  $P$ .
- 1676 5. Set  $j=\lceil n/u \rceil$ .
- 1677 6. For  $i=1, 2, \dots, j$ , do the following:
  - 1678 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  
1679 of that hash; etc.; continue in this fashion until a total of  $c$  hashes have been computed,  
1680 each on the result of the previous hash.
  - 1681 b. Concatenate copies of  $A_i$  to create a string  $B$  of length  $v$  bits (the final copy of  $A_i$  MAY be  
1682 truncated to create  $B$ ).
  - 1683 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  $I$   
1684 by setting  $I_j=(I_j+B) \bmod 2^v$  for each  $j$ . To perform this addition, treat each  $v$ -bit block as  
1685 a binary number represented most-significant bit first
- 1686 7. Concatenate  $A_1, A_2, \dots, A_j$  together to form a pseudo-random bit string,  $A$ .
- 1687 8. Use the first  $n$  bits of  $A$  as the output of this entire process

1688 When the password-based encryption mechanisms presented in this section are used to generate a key  
1689 and IV (if needed) from a password, salt, and an iteration count, the above algorithm is used. To  
1690 generate a key, the identifier byte  $ID$  is set to the value 1; to generate an IV, the identifier byte  $ID$  is set to  
1691 the value 2.

1692 When the password-based authentication mechanism presented in this section is used to generate a key  
1693 from a password, salt and an iteration count, the above algorithm is used. The identifier  $ID$  is set to the  
1694 value 3.

1695 **2.15.2 SHA-1-PBE for 128-bit RC4**

1696 SHA-1-PBE for 128-bit RC4, denoted **CKM\_PBE\_SHA1\_RC4\_128**, is a mechanism used for generating  
1697 a 128-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an  
1698 iteration count. The method used to generate the key is described above.

1699 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1700 key generation process. The parameter also has a field to hold the location of an application-supplied

1701 buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not  
1702 require an IV.  
1703 The key produced by this mechanism will typically be used for performing password-based encryption.

### 1704 **2.15.3 SHA-1\_PBE for 40-bit RC4**

1705 SHA-1-PBE for 40-bit RC4, denoted **CKM\_PBE\_SHA1\_RC4\_40**, is a mechanism used for generating a  
1706 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an  
1707 iteration count. The method used to generate the key is described above.  
1708 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1709 key generation process. The parameter also has a field to hold the location of an application-supplied  
1710 buffer which receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not  
1711 require an IV.  
1712 The key produced by this mechanism will typically be used for performing password-based encryption.

### 1713 **2.15.4 SHA-1\_PBE for 128-bit RC2-CBC**

1714 SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_128\_CBC**, is a mechanism used for  
1715 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest  
1716 algorithm and an iteration count. The method used to generate the key and IV is described above.  
1717 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1718 key generation process and the location of an application-supplied buffer which receives the 8-byte IV  
1719 generated by the mechanism.  
1720 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number  
1721 of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object  
1722 Identifier `pbeWithSHA1And128BitRC2-CBC`.  
1723 The key and IV produced by this mechanism will typically be used for performing password-based  
1724 encryption.

### 1725 **2.15.5 SHA-1\_PBE for 40-bit RC2-CBC**

1726 SHA-1-PBE for 40-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_40\_CBC**, is a mechanism used for  
1727 generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm  
1728 and an iteration count. The method used to generate the key and IV is described above.  
1729 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1730 key generation process and the location of an application-supplied buffer which receives the 8-byte IV  
1731 generated by the mechanism.  
1732 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number  
1733 of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object  
1734 Identifier `pbeWithSHA1And40BitRC2-CBC`.  
1735 The key and IV produced by this mechanism will typically be used for performing password-based  
1736 encryption.

## 1737 **2.16 RIPE-MD**

### 1738 **2.16.1 Definitions**

1739 Mechanisms:

- 1740 `CKM_RIPEMD128`
- 1741 `CKM_RIPEMD128_HMAC`
- 1742 `CKM_RIPEMD128_HMAC_GENERAL`
- 1743 `CKM_RIPEMD160`



1744 CKM\_RIPEMD160\_HMAC  
1745 CKM\_RIPEMD160\_HMAC\_GENERAL

## 1746 2.16.2 RIPE-MD 128 Digest

1747 The RIPE-MD 128 mechanism, denoted **CKM\_RIPEMD128**, is a mechanism for message digesting,  
1748 following the RIPE-MD 128 message-digest algorithm.

1749 It does not have a parameter.

1750 Constraints on the length of data are summarized in the following table:

1751 *Table 55, RIPE-MD 128: Data Length*

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

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

1752

## 1753 2.16.3 General-length RIPE-MD 128-HMAC

1754 The general-length RIPE-MD 128-HMAC mechanism, denoted **CKM\_RIPEMD128\_HMAC\_GENERAL**, is  
1755 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 128  
1756 hash function. The keys it uses are generic secret keys.

1757 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1758 output. This length should be in the range 0-16 (the output size of RIPE-MD 128 is 16 bytes). Signatures  
1759 (MACs) produced by this mechanism **MUST** be taken from the start of the full 16-byte HMAC output.

1760 *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
----------	----------------	-----	-------------------------------

## 1761 2.16.4 RIPE-MD 128-HMAC

1762 The RIPE-MD 128-HMAC mechanism, denoted **CKM\_RIPEMD128\_HMAC**, is a special case of the  
1763 general-length RIPE-MD 128-HMAC mechanism in Section 2.16.3.

1764 It has no parameter, and produces an output of length 16.

## 1765 2.16.5 RIPE-MD 160

1766 The RIPE-MD 160 mechanism, denoted **CKM\_RIPEMD160**, is a mechanism for message digesting,  
1767 following the RIPE-MD 160 message-digest defined in ISO-10118.

1768 It does not have a parameter.

1769 Constraints on the length of data are summarized in the following table:

1770 *Table 57, RIPE-MD 160: Data Length*

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

C_Digest	Any	20
----------	-----	----

1771 **2.16.6 General-length RIPE-MD 160-HMAC**

1772 The general-length RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC\_GENERAL**, is  
1773 a mechanism for signatures and verification. It uses the HMAC construction, based on the RIPE-MD 160  
1774 hash function. The keys it uses are generic secret keys.

1775 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1776 output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures  
1777 (MACs) produced by this mechanism MUST be taken from the start of the full 20-byte HMAC output.

1778 *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

1779 **2.16.7 RIPE-MD 160-HMAC**

1780 The RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC**, is a special case of the  
1781 general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.

1782 It has no parameter, and produces an output of length 20.

1783 **2.17 SET**

1784 **2.17.1 Definitions**

1785 Mechanisms:

1786 **CKM\_KEY\_WRAP\_SET\_OAEP**

1787 **2.17.2 SET mechanism parameters**

1788 **2.17.2.1 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS;**  
1789 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR**

1790 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** is a structure that provides the parameters to the  
1791 **CKM\_KEY\_WRAP\_SET\_OAEP** mechanism. It is defined as follows:

```

1792 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {
1793     CK_BYTE bBC;
1794     CK_BYTE_PTR pX;
1795     CK_ULONG ulXLen;
1796 } CK_KEY_WRAP_SET_OAEP_PARAMS;

```

1797 The fields of the structure have the following meanings:

- 1798 *bBC* block contents byte
- 1799 *pX* concatenation of hash of plaintext data (if present) and  
1800 extra data (if present)
- 1801 *ulXLen* length in bytes of concatenation of hash of plaintext data  
1802 (if present) and extra data (if present). 0 if neither is  
1803 present.

1804 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR** is a pointer to a  
1805 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS**.

## 1806 2.17.3 OAEP key wrapping for SET

1807 The OAEP key wrapping for SET mechanism, denoted **CKM\_KEY\_WRAP\_SET\_OAEP**, is a mechanism  
1808 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some  
1809 extra data MAY be wrapped together with the DES key. This mechanism is defined in the SET protocol  
1810 specifications.

1811 It takes a parameter, a **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure. This structure holds the  
1812 "Block Contents" byte of the data and the concatenation of the hash of plaintext data (if present) and the  
1813 extra data to be wrapped (if present). If neither the hash nor the extra data is present, this is indicated by  
1814 the *ulXLen* field having the value 0.

1815 When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present)  
1816 and the extra data (if present) is returned following the convention described [PKCS #11-Curr],  
1817 **Miscellaneous simple key derivation mechanisms**. Note that if the inputs to **C\_UnwrapKey** are such  
1818 that the extra data is not returned (e.g. the buffer supplied in the  
1819 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure is **NULL\_PTR**), then the unwrapped key object MUST  
1820 NOT be created, either.

1821 Be aware that when this mechanism is used to unwrap a key, the *bBC* and *pX* fields of the parameter  
1822 supplied to the mechanism MAY be modified.

1823 If an application uses **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP**, it may be preferable for it  
1824 simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data  
1825 (this concatenation MUST NOT be larger than 128 bytes), rather than calling **C\_UnwrapKey** twice. Each  
1826 call of **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP** requires an RSA decryption operation to be  
1827 performed, and this computational overhead MAY be avoided by this means.

## 1828 2.18 LYNKS

### 1829 2.18.1 Definitions

1830 Mechanisms:

1831 **CKM\_KEY\_WRAP\_LYNKS**

### 1832 2.18.2 LYNKS key wrapping

1833 The LYNKS key wrapping mechanism, denoted **CKM\_KEY\_WRAP\_LYNKS**, is a mechanism for  
1834 wrapping and unwrapping secret keys with DES keys. It MAY wrap any 8-byte secret key, and it produces  
1835 a 10-byte wrapped key, containing a cryptographic checksum.

1836 It does not have a parameter.

1837 To wrap an 8-byte secret key *K* with a DES key *W*, this mechanism performs the following steps:

- 1838 1. Initialize two 16-bit integers,  $sum_1$  and  $sum_2$ , to 0
- 1839 2. Loop through the bytes of *K* from first to last.
- 1840 3. Set  $sum_1 = sum_1 + \text{the key byte}$  (treat the key byte as a number in the range 0-255).
- 1841 4. Set  $sum_2 = sum_2 + sum_1$ .
- 1842 5. Encrypt *K* with *W* in ECB mode, obtaining an encrypted key, *E*.
- 1843 6. Concatenate the last 6 bytes of *E* with  $sum_2$ , representing  $sum_2$  most-significant bit first. The  
1844 result is an 8-byte block, *T*
- 1845 7. Encrypt *T* with *W* in ECB mode, obtaining an encrypted checksum, *C*.
- 1846 8. Concatenate *E* with the last 2 bytes of *C* to obtain the wrapped key.

1847 When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly,  
1848 an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity  
1849 bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.

1850

---

1851 **3 PKCS #11 Implementation Conformance**

1852 An implementation is a conforming implementation if it meets the conditions specified in one or more  
1853 server profiles specified in **[PKCS #11-Prof]**.

1854 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1855 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL  
1856 conform to all normative statements within the clauses specified for that profile and for any subclauses to  
1857 each of those clauses .

1858

---

## 1859 Appendix A. Acknowledgments

1860 The following individuals have participated in the creation of this specification and are gratefully  
1861 acknowledged:

1862

1863 **Participants:**

1864 Gil Abel, Athena Smartcard Solutions, Inc.

1865 Warren Armstrong, QuintessenceLabs

1866 Jeff Bartell, Semper Foris Solutions LLC

1867 Peter Bartok, Venafi, Inc.

1868 Anthony Berglas, Cryptsoft

1869 Joseph Brand, Semper Fortis Solutions LLC

1870 Kelley Burgin, National Security Agency

1871 Robert Burns, Thales e-Security

1872 Wan-Teh Chang, Google Inc.

1873 Hai-May Chao, Oracle

1874 Janice Cheng, Vormetric, Inc.

1875 Sangrae Cho, Electronics and Telecommunications Research Institute (ETRI)

1876 Doron Cohen, SafeNet, Inc.

1877 Fadi Cotran, Futurex

1878 Tony Cox, Cryptsoft

1879 Christopher Duane, EMC

1880 Chris Dunn, SafeNet, Inc.

1881 Valerie Fenwick, Oracle

1882 Terry Fletcher, SafeNet, Inc.

1883 Susan Gleeson, Oracle

1884 Sven Gossel, Charismathics

1885 John Green, QuintessenceLabs

1886 Robert Griffin, EMC

1887 Paul Grojean, Individual

1888 Peter Gutmann, Individual

1889 Dennis E. Hamilton, Individual

1890 Thomas Hardjono, M.I.T.

1891 Tim Hudson, Cryptsoft

1892 Gershon Janssen, Individual

1893 Seunghun Jin, Electronics and Telecommunications Research Institute (ETRI)

1894 Wang Jingman, Feitan Technologies

1895 Andrey Jivsov, Symantec Corp.

1896 Mark Joseph, P6R

1897 Stefan Kaesar, Infineon Technologies

1898 Greg Kazmierczak, Wave Systems Corp.

1899 Mark Knight, Thales e-Security  
1900 Darren Krahn, Google Inc.  
1901 Alex Krasnov, Infineon Technologies AG  
1902 Dina Kurktchi-Nimeh, Oracle  
1903 Mark Lambiase, SecureAuth Corporation  
1904 Lawrence Lee, GoTrust Technology Inc.  
1905 John Leiseboer, QuintessenceLabs  
1906 Sean Leon, Infineon Technologies  
1907 Geoffrey Li, Infineon Technologies  
1908 Howie Liu, Infineon Technologies  
1909 Hal Lockhart, Oracle  
1910 Robert Lockhart, Thales e-Security  
1911 Dale Moberg, Axway Software  
1912 Darren Moffat, Oracle  
1913 Valery Osheter, SafeNet, Inc.  
1914 Sean Parkinson, EMC  
1915 Rob Philpott, EMC  
1916 Mark Powers, Oracle  
1917 Ajai Puri, SafeNet, Inc.  
1918 Robert Relyea, Red Hat  
1919 Saikat Saha, Oracle  
1920 Subhash Sankuratipati, NetApp  
1921 Anthony Scarpino, Oracle  
1922 Johann Schoetz, Infineon Technologies AG  
1923 Rayees Shamsuddin, Wave Systems Corp.  
1924 Radhika Siravara, Oracle  
1925 Brian Smith, Mozilla Corporation  
1926 David Smith, Venafi, Inc.  
1927 Ryan Smith, Futurex  
1928 Jerry Smith, US Department of Defense (DoD)  
1929 Oscar So, Oracle  
1930 Graham Steel, Cryptosense  
1931 Michael Stevens, QuintessenceLabs  
1932 Michael StJohns, Individual  
1933 Jim Susoy, P6R  
1934 Sander Temme, Thales e-Security  
1935 Kiran Thota, VMware, Inc.  
1936 Walter-John Turnes, Gemini Security Solutions, Inc.  
1937 Stef Walter, Red Hat  
1938 James Wang, Vormetric  
1939 Jeff Webb, Dell  
1940 Peng Yu, Feitian Technologies

- 1941 Magda Zdunkiewicz, Cryptsoft
- 1942 Chris Zimman, Individual

1943

## Appendix B. Manifest constants

1944 The following constants have been defined for PKCS #11 V2.40. Also, refer to **[PKCS #11-Base]** and  
1945 **[PKCS #11-Curr]** for additional definitions.

```
1946 /*  
1947 * Copyright OASIS Open 2014. All rights reserved.  
1948 * OASIS trademark, IPR and other policies apply.  
1949 * http://www.oasis-open.org/policies-guidelines/ipr  
1950 */  
1951  
1952 #define CKK_KEA 0x00000005  
1953 #define CKK_RC2 0x00000011  
1954 #define CKK_RC4 0x00000012  
1955 #define CKK_DES 0x00000013  
1956 #define CKK_CAST 0x00000016  
1957 #define CKK_CAST3 0x00000017  
1958 #define CKK_CAST5 0x00000018  
1959 #define CKK_CAST128 0x00000018  
1960 #define CKK_RC5 0x00000019  
1961 #define CKK_IDEA 0x0000001A  
1962 #define CKK_SKIPJACK 0x0000001B  
1963 #define CKK_BATON 0x0000001C  
1964 #define CKK_JUNIPER 0x0000001D  
1965 #define CKM_MD2_RSA_PKCS 0x00000004  
1966 #define CKM_MD5_RSA_PKCS 0x00000005  
1967 #define CKM_RIPEMD128_RSA_PKCS 0x00000007  
1968 #define CKM_RIPEMD160_RSA_PKCS 0x00000008  
1969 #define CKM_RC2_KEY_GEN 0x00000100  
1970 #define CKM_RC2_ECB 0x00000101  
1971 #define CKM_RC2_CBC 0x00000102  
1972 #define CKM_RC2_MAC 0x00000103  
1973 #define CKM_RC2_MAC_GENERAL 0x00000104  
1974 #define CKM_RC2_CBC_PAD 0x00000105  
1975 #define CKM_RC4_KEY_GEN 0x00000110  
1976 #define CKM_RC4 0x00000111  
1977 #define CKM_DES_KEY_GEN 0x00000120  
1978 #define CKM_DES_ECB 0x00000121  
1979 #define CKM_DES_CBC 0x00000122  
1980 #define CKM_DES_MAC 0x00000123  
1981 #define CKM_DES_MAC_GENERAL 0x00000124  
1982 #define CKM_DES_CBC_PAD 0x00000125  
1983 #define CKM_MD2 0x00000200  
1984 #define CKM_MD2_HMAC 0x00000201  
1985 #define CKM_MD2_HMAC_GENERAL 0x00000202  
1986 #define CKM_MD5 0x00000210  
1987 #define CKM_MD5_HMAC 0x00000211  
1988 #define CKM_MD5_HMAC_GENERAL 0x00000212  
1989 #define CKM_RIPEMD128 0x00000230  
1990 #define CKM_RIPEMD128_HMAC 0x00000231  
1991 #define CKM_RIPEMD128_HMAC_GENERAL 0x00000232  
1992 #define CKM_RIPEMD160 0x00000240  
1993 #define CKM_RIPEMD160_HMAC 0x00000241  
1994 #define CKM_RIPEMD160_HMAC_GENERAL 0x00000242  
1995 #define CKM_CAST_KEY_GEN 0x00000300  
1996 #define CKM_CAST_ECB 0x00000301  
1997 #define CKM_CAST_CBC 0x00000302  
1998 #define CKM_CAST_MAC 0x00000303  
1999 #define CKM_CAST_MAC_GENERAL 0x00000304  
2000 #define CKM_CAST_CBC_PAD 0x00000305  
2001 #define CKM_CAST3_KEY_GEN 0x00000310
```



```

2002 #define CKM_CAST3_ECB 0x00000311
2003 #define CKM_CAST3_CBC 0x00000312
2004 #define CKM_CAST3_MAC 0x00000313
2005 #define CKM_CAST3_MAC_GENERAL 0x00000314
2006 #define CKM_CAST3_CBC_PAD 0x00000315
2007 #define CKM_CAST5_KEY_GEN 0x00000320
2008 #define CKM_CAST128_KEY_GEN 0x00000320
2009 #define CKM_CAST5_ECB 0x00000321
2010 #define CKM_CAST128_ECB 0x00000321
2011 #define CKM_CAST5_CBC 0x00000322
2012 #define CKM_CAST128_CBC 0x00000322
2013 #define CKM_CAST5_MAC 0x00000323
2014 #define CKM_CAST128_MAC 0x00000323
2015 #define CKM_CAST5_MAC_GENERAL 0x00000324
2016 #define CKM_CAST128_MAC_GENERAL 0x00000324
2017 #define CKM_CAST5_CBC_PAD 0x00000325
2018 #define CKM_CAST128_CBC_PAD 0x00000325
2019 #define CKM_RC5_KEY_GEN 0x00000330
2020 #define CKM_RC5_ECB 0x00000331
2021 #define CKM_RC5_CBC 0x00000332
2022 #define CKM_RC5_MAC 0x00000333
2023 #define CKM_RC5_MAC_GENERAL 0x00000334
2024 #define CKM_RC5_CBC_PAD 0x00000335
2025 #define CKM_IDEA_KEY_GEN 0x00000340
2026 #define CKM_IDEA_ECB 0x00000341
2027 #define CKM_IDEA_CBC 0x00000342
2028 #define CKM_IDEA_MAC 0x00000343
2029 #define CKM_IDEA_MAC_GENERAL 0x00000344
2030 #define CKM_IDEA_CBC_PAD 0x00000345
2031 #define CKM_MD5_KEY_DERIVATION 0x00000390
2032 #define CKM_MD2_KEY_DERIVATION 0x00000391
2033 #define CKM_PBE_MD2_DES_CBC 0x000003A0
2034 #define CKM_PBE_MD5_DES_CBC 0x000003A1
2035 #define CKM_PBE_MD5_CAST_CBC 0x000003A2
2036 #define CKM_PBE_MD5_CAST3_CBC 0x000003A3
2037 #define CKM_PBE_MD5_CAST5_CBC 0x000003A4
2038 #define CKM_PBE_MD5_CAST128_CBC 0x000003A4
2039 #define CKM_PBE_SHA1_CAST5_CBC 0x000003A5
2040 #define CKM_PBE_SHA1_CAST128_CBC 0x000003A5
2041 #define CKM_PBE_SHA1_RC4_128 0x000003A6
2042 #define CKM_PBE_SHA1_RC4_40 0x000003A7
2043 #define CKM_PBE_SHA1_RC2_128_CBC 0x000003AA
2044 #define CKM_PBE_SHA1_RC2_40_CBC 0x000003AB
2045 #define CKM_KEY_WRAP_LYNKS 0x00000400
2046 #define CKM_KEY_WRAP_SET_OAEP 0x00000401
2047 #define CKM_SKIPJACK_KEY_GEN 0x00001000
2048 #define CKM_SKIPJACK_ECB64 0x00001001
2049 #define CKM_SKIPJACK_CBC64 0x00001002
2050 #define CKM_SKIPJACK_OFB64 0x00001003
2051 #define CKM_SKIPJACK_CFB64 0x00001004
2052 #define CKM_SKIPJACK_CFB32 0x00001005
2053 #define CKM_SKIPJACK_CFB16 0x00001006
2054 #define CKM_SKIPJACK_CFB8 0x00001007
2055 #define CKM_SKIPJACK_WRAP 0x00001008
2056 #define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009
2057 #define CKM_SKIPJACK_RELAYX 0x0000100a
2058 #define CKM_KEA_KEY_PAIR_GEN 0x00001010
2059 #define CKM_KEA_KEY_DERIVE 0x00001011
2060 #define CKM_FORTEZZA_TIMESTAMP 0x00001020
2061 #define CKM_BATON_KEY_GEN 0x00001030
2062 #define CKM_BATON_ECB128 0x00001031
2063 #define CKM_BATON_ECB96 0x00001032
2064 #define CKM_BATON_CBC128 0x00001033
2065 #define CKM_BATON_COUNTER 0x00001034

```

```
2066 #define CKM_BATON_SHUFFLE 0x00001035
2067 #define CKM_BATON_WRAP 0x00001036
2068 #define CKM_JUNIPER_KEY_GEN 0x00001060
2069 #define CKM_JUNIPER_ECB128 0x00001061
2070 #define CKM_JUNIPER_CBC128 0x00001062
2071 #define CKM_JUNIPER_COUNTER 0x00001063
2072 #define CKM_JUNIPER_SHUFFLE 0x00001064
2073 #define CKM_JUNIPER_WRAP 0x00001065
2074 #define CKM_FASTHASH 0x00001070
```

2075

2076

## Appendix C. Revision History

2077

Revision	Date	Editor	Changes Made
wd01	May 16, 2013	Susan Gleeson	Initial Template import
wd02	July 7, 2013	Susan Gleeson	Fix references, add participants list, minor cleanup
wd03	October 27, 2013	Robert Griffin	Final participant list and other editorial changes for Committee Specification Draft
csd01	October 30, 2013	OASIS	Committee Specification Draft
wd04	February 19, 2014	Susan Gleeson	Incorporate changes from v2.40 public review
wd05	February 20, 2014	Susan Gleeson	Regenerate table of contents (oversight from wd04)
WD06	February 21, 2014	Susan Gleeson	Remove CKM_PKCS5_PBKD2 from the mechanisms in Table 1.
csd02	April 23, 2014	OASIS	Committee Specification Draft
csd02a	Sep 3 2013 <sup>34</sup>	Robert Griffin	Updated revision history and participant list in preparation for Committee Specification ballot
<a href="#">wd07</a>	<a href="#">Nov 3 2014</a>	<a href="#">Robert Griffin</a>	<a href="#">Editorial corrections</a>

2078