

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

## Committee Specification Draft 01

30 October 2013

### Specification URIs

#### This version:

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd01/pkcs11-hist-v2.40-csd01.doc>

(Authoritative)

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd01/pkcs11-hist-v2.40-csd01.html>

<http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd01/pkcs11-hist-v2.40-csd01.pdf>

#### Previous version:

N/A

#### Latest version:

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

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

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

#### Technical Committee:

OASIS PKCS 11 TC

#### Chairs:

Robert Griffin ([robert.griffin@rsa.com](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 ([czimman@bloomberg.com](mailto:czimman@bloomberg.com)), Bloomberg Finance L.P.

#### Related work:

This specification is related to:

- *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40*. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40*. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- *PKCS #11 Cryptographic Token Interface Profiles Version 2.40*. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html>.

#### Abstract:

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

**Status:**

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

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

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

**Citation format:**

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

**[PKCS11-hist]**

*PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 2.40*. 30 October 2013. OASIS Committee Specification Draft 01. <http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/csd01/pkcs11-hist-v2.40-csd01.html>.

---

# Notices

Copyright © OASIS Open 2013. All Rights Reserved.

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

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

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

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

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

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

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

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

---

# Table of Contents

1	Introduction .....	8
1.1	Terminology .....	8
1.2	Definitions .....	8
1.3	Normative References .....	9
1.4	Non-Normative References .....	9
2	Mechanisms .....	13
2.1	FORTEZZA timestamp .....	16
2.2	KEA .....	16
2.2.1	Definitions .....	16
2.2.2	KEA mechanism parameters .....	16
2.2.3	KEA public key objects .....	17
2.2.4	KEA private key objects .....	18
2.2.5	KEA key pair generation .....	18
2.2.6	KEA key derivation .....	19
2.3	RC2 .....	20
2.3.1	Definitions .....	20
2.3.2	RC2 secret key objects .....	20
2.3.3	RC2 mechanism parameters .....	21
2.3.4	RC2 key generation .....	22
2.3.5	RC2-ECB .....	22
2.3.6	RC2-CBC .....	23
2.3.7	RC2-CBC with PKCS padding .....	23
2.3.8	General-length RC2-MAC .....	24
2.3.9	RC2-MAC .....	24
2.4	RC4 .....	25
2.4.1	Definitions .....	25
2.4.2	RC4 secret key objects .....	25
2.4.3	RC4 key generation .....	25
2.4.4	RC4 mechanism .....	26
2.5	RC5 .....	26
2.5.1	Definitions .....	26
2.5.2	RC5 secret key objects .....	26
2.5.3	RC5 mechanism parameters .....	27
2.5.4	RC5 key generation .....	28
2.5.5	RC5-ECB .....	28
2.5.6	RC5-CBC .....	29
2.5.7	RC5-CBC with PKCS padding .....	29
2.5.8	General-length RC5-MAC .....	30
2.5.9	RC5-MAC .....	30
2.6	General block cipher .....	31
2.6.1	Definitions .....	31
2.6.2	DES secret key objects .....	32
2.6.3	CAST secret key objects .....	33

2.6.4 CAST3 secret key objects .....	33
2.6.5 CAST128 (CAST5) secret key objects .....	34
2.6.6 IDEA secret key objects .....	34
2.6.7 CDMF secret key objects .....	35
2.6.8 General block cipher mechanism parameters.....	35
2.6.9 General block cipher key generation.....	35
2.6.10 General block cipher ECB.....	36
2.6.11 General block cipher CBC.....	36
2.6.12 General block cipher CBC with PKCS padding.....	37
2.6.13 General-length general block cipher MAC .....	38
2.6.14 General block cipher MAC .....	38
2.7 SKIPJACK.....	39
2.7.1 Definitions.....	39
2.7.2 SKIPJACK secret key objects .....	39
2.7.3 SKIPJACK Mechanism parameters .....	40
2.7.4 SKIPJACK key generation .....	42
2.7.5 SKIPJACK-ECB64 .....	42
2.7.6 SKIPJACK-CBC64 .....	42
2.7.7 SKIPJACK-OFB64 .....	42
2.7.8 SKIPJACK-CFB64.....	43
2.7.9 SKIPJACK-CFB32.....	43
2.7.10 SKIPJACK-CFB16.....	43
2.7.11 SKIPJACK-CFB8.....	44
2.7.12 SKIPJACK-WRAP .....	44
2.7.13 SKIPJACK-PRIVATE-WRAP .....	44
2.7.14 SKIPJACK-RELAYX.....	44
2.8 BATON.....	44
2.8.1 Definitions.....	44
2.8.2 BATON secret key objects .....	45
2.8.3 BATON key generation .....	45
2.8.4 BATON-ECB128 .....	46
2.8.5 BATON-ECB96.....	46
2.8.6 BATON-CBC128 .....	46
2.8.7 BATON-COUNTER .....	47
2.8.8 BATON-SHUFFLE .....	47
2.8.9 BATON WRAP .....	47
2.9 JUNIPER.....	47
2.9.1 Definitions.....	47
2.9.2 JUNIPER secret key objects .....	48
2.9.3 JUNIPER key generation .....	48
2.9.4 JUNIPER-ECB128 .....	49
2.9.5 JUNIPER-CBC128 .....	49
2.9.6 JUNIPER-COUNTER .....	49
2.9.7 JUNIPER-SHUFFLE .....	49
2.9.8 JUNIPER WRAP .....	50

2.10 MD2 .....	50
2.10.1 Definitions .....	50
2.10.2 MD2 digest .....	50
2.10.3 General-length MD2-HMAC .....	50
2.10.4 MD2-HMAC .....	51
2.10.5 MD2 key derivation .....	51
2.11 MD5 .....	51
2.11.1 Definitions .....	51
2.11.2 MD5 Digest .....	52
2.11.3 General-length MD5-HMAC .....	52
2.11.4 MD5-HMAC .....	52
2.11.5 MD5 key derivation .....	52
2.12 FASTHASH .....	53
2.12.1 Definitions .....	53
2.12.2 FASTHASH digest .....	53
2.13 PKCS #5 and PKCS #5-style password-based encryption (PBD) .....	53
2.13.1 Definitions .....	53
2.13.2 Password-based encryption/authentication mechanism parameters .....	54
2.13.3 MD2-PBE for DES-CBC .....	54
2.13.4 MD5-PBE for DES-CBC .....	54
2.13.5 MD5-PBE for CAST-CBC .....	55
2.13.6 MD5-PBE for CAST3-CBC .....	55
2.13.7 MD5-PBE for CAST128-CBC (CAST5-CBC) .....	55
2.13.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC) .....	55
2.14 PKCS #12 password-based encryption/authentication mechanisms .....	56
2.14.1 SHA-1-PBE for 128-bit RC4 .....	56
2.14.2 SHA-1_PBE for 40-bit RC4 .....	57
2.14.3 SHA-1_PBE for 128-bit RC2-CBC .....	57
2.14.4 SHA-1_PBE for 40-bit RC2-CBC .....	57
2.15 RIPE-MD .....	57
2.15.1 Definitions .....	57
2.15.2 RIPE-MD 128 Digest .....	58
2.15.3 General-length RIPE-MD 128-HMAC .....	58
2.15.4 RIPE-MD 128-HMAC .....	58
2.15.5 RIPE-MD 160 .....	58
2.15.6 General-length RIPE-MD 160-HMAC .....	58
2.15.7 RIPE-MD 160-HMAC .....	59
2.16 SET .....	59
2.16.1 Definitions .....	59
2.16.2 SET mechanism parameters .....	59
2.16.3 OAEP key wrapping for SET .....	59
2.17 LYNKS .....	60
2.17.1 Definitions .....	60
2.17.2 LYNKS key wrapping .....	60
3 PKCS #11 Implementation Conformance .....	61

Appendix A.	Acknowledgments .....	62
Appendix B.	Manifest constants .....	64
Appendix C.	Revision History .....	67

---

# 1 Introduction

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

All text is normative unless otherwise labeled.

## 1.1 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 **[PKCS #11-Base]** *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html>.

**[PKCS #11-Curr]** *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40*. Latest version. <http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html>.

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

**[RFC2119]**.

## 1.2 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 2630)
<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.



41	<b>MAC</b>	Message Authentication Code
42	<b>MD2</b>	RSA Security's MD2 message-digest algorithm, as defined in RFC
43		1319.
44	<b>MD5</b>	RSA Security's MD5 message-digest algorithm, as defined in RFC
45		1321.
46	<b>PRF</b>	Pseudo random function.
47	<b>RSA</b>	The RSA public-key cryptosystem.
48	<b>RC2</b>	RSA Security's RC2 symmetric block cipher.
49	<b>RC4</b>	RSA Security's proprietary RC4 symmetric stream cipher.
50	<b>RC5</b>	RSA Security's RC5 symmetric block cipher.
51	<b>SET</b>	<b>The Secure Electronic Transaction protocol.</b>
52	<b>SHA-1</b>	The (revised) Secure Hash Algorithm with a 160-bit message digest,
53		as defined in FIPS PUB 180-2.
54	<b>SKIPJACK</b>	MISSI's SKIPJACK block cipher.
55	<b>UTF-8</b>	Universal Character Set (UCS) transformation format (UTF) that
56		represents ISO 10646 and UNICODE strings with a variable number
57		of octets

### 58 1.3 Normative References

59	<b>[PKCS #11-Base]</b>	<i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i> Latest
60		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-</a>
61		<a href="http://docs.oasis-open.org/pkcs11/pkcs11-base-v2.40/pkcs11-base-v2.40.html">v2.40.html</a> .
62	<b>[PKCS #11-Curr]</b>	<i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification</i>
63		<i>Version 2.40.</i> 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-</a>
64		<a href="http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html">curr/v2.40/pkcs11-curr-v2.40.html</a> .
65	<b>[PKCS #11-Prof]</b>	<i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i> Latest version.
66		<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-</a>
67		<a href="http://docs.oasis-open.org/pkcs11/pkcs11-profiles-v2.40/pkcs11-profiles-v2.40.html">v2.40.html</a> .
68	<b>[RFC2119]</b>	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP
69		14, RFC 2119, March 1997. <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> .

### 70 1.4 Non-Normative References

71	<b>[ANSI C]</b>	ANSI/ISO. <i>American National Standard for Programming Languages – C.</i> 1990
72	<b>[ANSI X9.31]</b>	Accredited Standards Committee X9. <i>Digital Signatures Using Reversible Public</i>
73		<i>Key Cryptography for the Financial Services Industry (rDSA).</i> 1998.
74	<b>[ANSI X9.42]</b>	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial</i>
75		<i>Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm</i>
76		<i>Cryptography.</i> 2003
77	<b>[ANSI X9.62]</b>	Accredited Standards Committee X9. <i>Public Key Cryptography for the Financial</i>
78		<i>Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).</i> 1998
79	<b>[CC/PP]</b>	W3C. <i>Composite Capability/Preference Profiles (CC/PP): Structure and</i>
80		<i>Vocabularies.</i> World Wide Web Consortium, January 2004. URL:
81		<a href="http://www.w3.org/RT/CCPP-struct-vocab/">http://www.w3.org/RT/CCPP-struct-vocab/</a>
82	<b>[CDPD]</b>	Ameritech Mobile Communications et al. <i>Cellular Digital Packet Data System</i>
83		<i>Specifications: Part 406: Airlink Security.</i> 1993
84	<b>[FIPS PUB 46-3]</b>	NIST. <i>FIPS 46-3: Data Encryption Standard (DES).</i> October 26, 2999. URL:
85		<a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>

86	<b>[FIPS PUB 74]</b>	NIST. <i>FIPS 74: Guidelines for Implementing and Using the NBS Data Encryption Standard</i> . April 1, 1981. URL: <a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
87		
88	<b>[FIPS PUB 81]</b>	NIST. <i>FIPS 81: DES Modes of Operation</i> . December 1980. URL:
89		<a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
90	<b>[FIPS PUB 113]</b>	NIST. <i>FIPS 113: Computer Data Authentication</i> . May 30, 1985. URL:
91		<a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
92	<b>[FIPS PUB 180-2]</b>	NIST. <i>FIPS 180-2: Secure Hash Standard</i> . August 1, 2002. URL:
93		<a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
94	<b>[FIPS PUB 186-2]</b>	NIST. <i>FIPS 186-2: Digital Signature Standard</i> . January 27, 2000. URL:
95		<a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
96	<b>[FIPS PUB 197]</b>	NIST. <i>FIPS 197: Advanced Encryption Standard (AES)</i> . November 26, 2001.
97		URL: <a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>
98	<b>[FORTEZZA CIPG]</b>	NSA, Workstation Security Products. <i>FORTEZZA Cryptologic Interface</i>
99		<i>Programmers Guide, Revision 1.52</i> . November 1985
100	<b>[GCS-API]</b>	X/Open Company Ltd. <i>Generic Cryptographic Service API (GCS-API), Base –</i>
101		<i>Draft 2</i> . February 14, 1995.
102	<b>[ISO/IEC 7816-1]</b>	ISO. <i>Information Technology – Identification Cards – Integrated Circuit(s) with</i>
103		<i>Contacts – Part 1: Physical Characteristics</i> . 1998.
104	<b>[ISO/IEC 7816-4]</b>	ISO. <i>Information Technology – Identification Cards – Integrated Circuit(s) with</i>
105		<i>Contacts – Part 4: Interindustry Commands for Interchange</i> . 1995.
106	<b>[ISO/IEC 8824-1]</b>	ISO. <i>Information Technology – Abstract Syntax Notation One (ASN.1):</i>
107		<i>Specification of Base Notation</i> . 2002.
108	<b>[ISO/IEC 8825-1]</b>	ISO. <i>Information Technology – ASN.1 Encoding Rules: Specification of Basic</i>
109		<i>Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished</i>
110		<i>Encoding Rules (DER)</i> . 2002.
111	<b>[ISO/IEC 9594-1]</b>	ISO. <i>Information Technology – Open System Interconnection – The Directory:</i>
112		<i>Overview of Concepts, Models and Services</i> . 2001.
113	<b>[ISO/IEC 9594-8]</b>	ISO. <i>Information Technology – Open Systems Interconnection – The Directory:</i>
114		<i>Public-key and Attribute Certificate Frameworks</i> . 2001.
115	<b>[ISO/IEC 9796-2]</b>	ISO. <i>Information Technology – Security Techniques – Digital Signature Scheme</i>
116		<i>Giving Message Recovery – Part 2: Integer factorization based mechanisms</i> .
117		2002.
118	<b>[Java MIDP]</b>	Java Community Process. <i>Mobile Information Device Profile for Java 2 Micro</i>
119		<i>Edition</i> . November 2002. URL: <a href="http://jcp.org/jsr/detail/118.jsp">http://jcp.org/jsr/detail/118.jsp</a>
120	<b>[MeT-PTD]</b>	MeT. <i>MeT PTD Definition – Personal Trusted Device Definition, Version 1.0</i> .
121		February 2003. URL: <a href="http://www.mobiletransaction.org">http://www.mobiletransaction.org</a>
122	<b>[PCMCIA]</b>	Personal Computer Memory Card International Association. <i>PC Card Standard,</i>
123		<i>Release 2.1</i> . July 1993.
124	<b>[PKCS #1]</b>	RSA Laboratories. <i>RSA Cryptography Standard, v2.1</i> . June 14, 2002
125	<b>[PKCS #3]</b>	RSA Laboratories. <i>Diffie-Hellman Key-Agreement Standard, v1.4</i> . November
126		1993.
127	<b>[PKCS #5]</b>	RSA Laboratories. <i>Password-Based Encryption Standard, v2.0</i> . March 26,
128		1999.
129	<b>[PKCS #7]</b>	RSA Laboratories. <i>Cryptographic Message Syntax Standard, v1.5</i> . November
130		1993
131	<b>[PKCS #8]</b>	RSA Laboratories. <i>Private-Key Information Syntax Standard, v1.2</i> . November
132		1993.
133	<b>[PKCS #11-UG]</b>	<i>PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40</i> . Latest
134		version. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-</a>
135		<a href="http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html">v2.40.html</a> .
136	<b>[PKCS #11-C]</b>	RSA Laboratories. <i>PKCS#11: Conformance Profile Specification</i> . October
137		2000.
138	<b>[PKCS #11-P]</b>	RSA Laboratories. <i>PKCS #11 Profiles for mobile devices</i> . June 2003.

<b>[PKCS #12]</b>	RSA Laboratories. <i>Personal Information Exchange Syntax Standard</i> , v1.0. June 1999.
<b>[RFC 1319]</b>	B. Kaliski. <i>RFC 1319: The MD2 Message-Digest Algorithm</i> . RSA Laboratories, April 1992. URL: <a href="http://ietf.org/rfc/rfc1319.txt">http://ietf.org/rfc/rfc1319.txt</a>
<b>[RFC 1321]</b>	R. Rivest. <i>RFC 1321: The MD5 Message-Digest Algorithm</i> . MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. URL: <a href="http://ietf.org/rfc/rfc1321.txt">http://ietf.org/rfc/rfc1321.txt</a>
<b>[RFC 1421]</b>	J. Linn. <i>RFC 1421: Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures</i> . IAB IRTF PSRG, IETF PEM WG, February 1993. URL: <a href="http://ietf.org/rfc/rfc1421.txt">http://ietf.org/rfc/rfc1421.txt</a>
<b>[RFC 2045]</b>	Freed, N., and Borenstein. <i>RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies</i> . November 1996. URL: <a href="http://ietf.org/rfc/rfc2045.txt">http://ietf.org/rfc/rfc2045.txt</a>
<b>[RFC 2246]</b>	T. Dierks and C. Allen. <i>RFC 2246: The TLS Protocol Version 1.0</i> . Certicom, January 1999. URL: <a href="http://ietf.org/rfc/rfc2246.txt">http://ietf.org/rfc/rfc2246.txt</a>
<b>[RFC 2279]</b>	F. Yergeau. <i>RFC 2279: UTF-8, a transformation format of ISO 10646</i> . Alis Technologies, January 1998. URL: <a href="http://ietf.org/rfc/rfc2279.txt">http://ietf.org/rfc/rfc2279.txt</a>
<b>[RFC 2534]</b>	Masinter, L., Wing, D., Mutz, A., and K. Holtman. <i>RFC 2534: Media Features for Display, Print and Fax</i> . March 1999. URL: <a href="http://ietf.org/rfc/rfc2534.txt">http://ietf.org/rfc/rfc2534.txt</a>
<b>[RFC 2630]</b>	R. Houseley. <i>RFC 2630: cryptographic Message Syntax</i> . June 1999. URL: <a href="http://ietf.org/rfc/rfc2630.txt">http://ietf.org/rfc/rfc2630.txt</a>
<b>[RFC 2743]</b>	J. Linn. <i>RFC 2743: Generic Security Service Application Program Interface Version 2, Update 1</i> . RSA Laboratories, January 2000. URL: <a href="http://ietf.org/rfc/rfc2743.txt">http://ietf.org/rfc/rfc2743.txt</a>
<b>[RFC 2744]</b>	J. Wray. <i>RFC 2744: Generic Security Services API Version 2: C-bindings</i> . Iris Associates, January 2000. URL: <a href="http://ietf.org/rfc/rfc2744.txt">http://ietf.org/rfc/rfc2744.txt</a>
<b>[SEC-1]</b>	Standards for Efficient Cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography</i> . Version 1.0, September 20, 2000.
<b>[SEC-2]</b>	Standards for Efficient cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters</i> . Version 1.0, September 20, 2000.
<b>[TLS]</b>	IETF. <i>RFC 2246: The TLS Protocol Version 1.0</i> . January 1999. URL: <a href="http://ietf.org/rfc/rfc2256.txt">http://ietf.org/rfc/rfc2256.txt</a>
<b>[WIM]</b>	WAP. <i>Wireless Identity Module</i> . – WAP-260-WIP-20010712.a. July 2001. URL: <a href="http://www.wapforum.org">http://www.wapforum.org</a>
<b>[WPKI]</b>	WAP. <i>Wireless PKI</i> . – WAP-217-WPKI-20010424-a. April 2001. URL: <a href="http://www.wapforum.org">http://www.wapforum.org</a>
<b>[WTLS]</b>	WAP. <i>Wireless Transport Layer Security Version</i> – WAP-261-WTLS-20010406-a. April 2001. URL: <a href="http://www.wapforum.org">http://www.wapforum.org</a>
<b>[X.500]</b>	ITU-T. <i>Information Technology – Open Systems Interconnection – The Directory: Overview of Concepts, Models and Services</i> . February 2001. (Identical to ISO/IEC 9594-1)
<b>[X.509]</b>	ITU-T. <i>Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks</i> . March 2000. (Identical to ISO/IEC 9594-8)
<b>[X.680]</b>	ITU-T. <i>Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation</i> . July 2002. (Identical to ISO/IEC 8824-1)
<b>[X.690]</b>	ITU-T. <i>Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)</i> . July 2002. (Identical to ISO/IEC 8825-1)



## 2 Mechanisms

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

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

Table 1, Mechanisms vs. Functions

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_KEY_PAIR_GEN					X		
CKM_KEY_DERIVE							X
CKM_RC2_KEY_GEN					X		
CKM_RC2_ECB	X					X	
CKM_RC2_CBC	X					X	
CKM_RC2_CBC_PAD	X					X	
CKM_RC2_MAC_GENERAL		X					
CKM_RC2_MAC		X					
CKM_RC4_KEY_GEN					X		
CKM_RC4	X						
CKM_RC5_KEY_GEN					X		
CKM_RC5_ECB	X					X	
CKM_RC5_CBC	X					X	
CKM_RC5_CBC_PAD	X					X	
CKM_RC5_MAC_GENERAL		X					
CKM_RC5_MAC		X					
CKM_DES_KEY_GEN					X		
CKM_DES_ECB	X					X	
CKM_DES_CBC	X					X	
CKM_DES_CBC_PAD	X					X	
CKM_DES_MAC_GENERAL		X					
CKM_DES_MAC		X					
CKM_CAST_KEY_GEN					X		
CKM_CAST_ECB	X					X	
CKM_CAST_CBC	X					X	
CKM_CAST_CBC_PAD	X					X	
CKM_CAST_MAC_GENERAL		X					
CKM_CAST_MAC		X					
CKM_CAST3_KEY_GEN					X		

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR <sup>1</sup>	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
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_OF64	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						
CKM_BATON_CBC128	X						
CKM_BATON_COUNTER	X						
CKM_BATON_SHUFFLE	X						

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

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

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

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

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

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

## 2.1 FORTEZZA timestamp

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

**It has no parameters.**

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

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

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

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

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

## 2.2 KEA

### 2.2.1 Definitions

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

Mechanisms:

CKM\_KEA\_KEY\_PAIR\_GEN

CKM\_KEA\_KEY\_DERIVE

### 2.2.2 KEA mechanism parameters

#### 2.2.2.1 CK\_KEA\_DERIVE\_PARAMS; CK\_KEA\_DERIVE\_PARAMS\_PTR

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

```
typedef struct CK_KEA_DERIVE_PARAMS {
    CK_BBOOL isSender;
    CK_ULONG ulRandomLen;
    CK_BYTE_PTR pRandomA;
    CK_BYTE_PTR pRandomB;
    CK_ULONG ulPublicDataLen;
    CK_BYTE_PTR pPublicData;
} CK_KEA_DERIVE_PARAMS;
```

The fields of the structure have the following meanings:



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

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

251                    *pRandomA*      pointer to Ra data

252                    *pRandomB*      pointer to Rb data

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

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

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

## 256    2.2.3 KEA public key objects

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

260    *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 $y$

261    <sup>1</sup> Refer to [PKCS #11-Base] table 15 for footnotes

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

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

```

265    CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
266    CK_KEY_TYPE keyType = CKK_KEA;
267    CK_UTF8CHAR label[] = "A KEA public key object";
268    CK_BYTE prime[] = {...};
269    CK_BYTE subprime[] = {...};
270    CK_BYTE base[] = {...};
271    CK_BYTE value[] = {...};
272    CK_ATTRIBUTE template[] = {
273       {CKA_CLASS, &class, sizeof(class)},
274       {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
275       {CKA_TOKEN, &true, sizeof(true)},
276       {CKA_LABEL, label, sizeof(label)-1},
277       {CKA_PRIME, prime, sizeof(prime)},
278       {CKA_SUBPRIME, subprime, sizeof(subprime)},
279       {CKA_BASE, base, sizeof(base)},
280       {CKA_VALUE, value, sizeof(value)}
281    };

```

282

## 2.2.4 KEA private key objects

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

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 $x$

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

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

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

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

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_KEA;
CK_UTF8CHAR label[] = "A KEA private key object";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)}, Algorithm, as defined by NISTS
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label) - 1},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_ID, id, sizeof(id)},
    {CKA_SENSITIVE, &true, sizeof(true)},
    {CKA_DERIVE, &true, sizeof(true)},
    {CKA_PRIME, prime, sizeof(prime)},
    {CKA_SUBPRIME, subprime, sizeof(subprime)},
    {CKA_BASE, base, sizeof(base)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.2.5 KEA key pair generation

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

It does not have a parameter.

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

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

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

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

## 2.2.6 KEA key derivation

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

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

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

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

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

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.

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

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

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

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

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

## 2.3 RC2

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

### 2.3.1 Definitions

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

Mechanisms:

CKM\_RC2\_KEY\_GEN  
CKM\_RC2\_ECB  
CKM\_RC2\_CBC  
CKM\_RC2\_MAC  
CKM\_RC2\_MAC\_GENERAL  
CKM\_RC2\_CBC\_PAD

### 2.3.2 RC2 secret key objects

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

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

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

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC2;
CK_UTF8CHAR label[] = "An RC2 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.3.3 RC2 mechanism parameters

### 2.3.3.1 CK\_RC2\_PARAMS; CK\_RC2\_PARAMS\_PTR

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

```
typedef CK_ULONG CK_RC2_PARAMS;
```

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

### 2.3.3.2 CK\_RC2\_CBC\_PARAMS; CK\_RC2\_CBC\_PARAMS\_PTR

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

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

The fields of the structure have the following meanings:

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

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

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

### 2.3.3.3 CK\_RC2\_MAC\_GENERAL\_PARAMS; CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR

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

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

The fields of the structure have the following meanings:

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

*ulMacLength* length of the MAC produced, in bytes

**CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR** is a pointer to a **CK\_RC2\_MAC\_GENERAL\_PARAMS**.

## 2.3.4 RC2 key generation

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

It does not have a parameter.

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

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

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

## 2.3.5 RC2-ECB

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

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

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

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

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

Table 7 RC2-ECB: Key and Data Length

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



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

### 2.3.6 RC2-CBC

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

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

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

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

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

Table 8, RC2-CBC: Key and Data Length

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

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

### 2.3.7 RC2-CBC with PKCS padding

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

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

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

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

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

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

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

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

### 2.3.8 General-length RC2-MAC

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

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

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

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

Table 10, General-length RC2-MAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	RC2	Any	0-8, as specified in parameters
C_Verify	RC2	Any	0-8, as specified in parameters

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

### 2.3.9 RC2-MAC

RC2-MAC, denoted by **CKM\_RC2\_MAC**, is a special case of the general-length RC2-MAC mechanism (see Section 2.3.8). Instead of taking a **CK\_RC2\_MAC\_GENERAL\_PARAMS** parameter, it takes a **CK\_RC2\_PARAMS** parameter, which only contains the effective number of bits in the RC2 search space. RC2-MAC always produces and verifies 4-byte MACs.

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

Table 11, RC2-MAC: Key and Data Length

Function	Key type	Data length	Signature length
C_Sign	RC2	Any	4
C_Verify	RC2	Any	4

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



## 2.4 RC4

### 2.4.1 Definitions

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

Mechanisms

CKM\_RC4\_KEY\_GEN

CKM\_RC4

### 2.4.2 RC4 secret key objects

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

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

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

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC4;
CK_UTF8CHAR label[] = "An RC4 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

### 2.4.3 RC4 key generation

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

It does not have a parameter.

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

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

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

## 2.4.4 RC4 mechanism

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

It does not have a parameter.

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

Table 13, RC4: Key and Data Length

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

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

## 2.5 RC5

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

### 2.5.1 Definitions

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

Mechanisms:

CKM\_RC5\_KEY\_GEN  
CKM\_RC5\_ECB  
CKM\_RC5\_CBC  
CKM\_RC5\_MAC  
CKM\_RC5\_MAC\_GENERAL  
CMK\_RC5\_CBC\_PAD

### 2.5.2 RC5 secret key objects

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

Table 14, RC5 Secret Key Object

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

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

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

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

```

605 CK_ATTRIBUTE template[] = {
606     {CKA_CLASS, &class, sizeof(class)},
607     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
608     {CKA_TOKEN, &true, sizeof(true)},
609     {CKA_LABEL, label, sizeof(label)-1},
610     {CKA_ENCRYPT, &true, sizeof(true)},
611     {CKA_VALUE, value, sizeof(value)}
612 };

```

## 613 2.5.3 RC5 mechanism parameters

### 614 2.5.3.1 CK\_RC5\_PARAMS; CK\_RC5\_PARAMS\_PTR

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

```

617 typedef struct CK_RC5_PARAMS {
618     CK_ULONG ulWordsize;
619     CK_ULONG ulRounds;
620 } CK_RC5_PARAMS;

```

621 The fields of the structure have the following meanings:

622 *ulWordsize*      wordsize of RC5 cipher in bytes

623 *ulRounds*      number of rounds of RC5 encipherment

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

### 625 2.5.3.2 CK\_RC5\_CBC\_PARAMS; CK\_RC5\_CBC\_PARAMS\_PTR

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

```

628 typedef struct CK_RC5_CBC_PARAMS {
629     CK_ULONG ulWordsize;
630     CK_ULONG ulRounds;
631     CK_BYTE_PTR pIv;
632     CK_ULONG ulIvLen;
633 } CK_RC5_CBC_PARAMS;

```

634 The fields of the structure have the following meanings:

635 *ulwordSize*      wordsize of RC5 cipher in bytes

636 *ulRounds*      number of rounds of RC5 encipherment

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

638 *ulIVLen*      length of initialization vector (must be same as  
639                      blocksize)

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

### 641 2.5.3.3 CK\_RC5\_MAC\_GENERAL\_PARAMS; 642 CK\_RC5\_MAC\_GENERAL\_PARAMS\_PTR

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

```

645 typedef struct CK_RC5_MAC_GENERAL_PARAMS {
646     CK_ULONG ulWordsize;
647     CK_ULONG ulRounds;
648     CK_ULONG ulMacLength;
649 } CK_RC5_MAC_GENERAL_PARAMS;

```

650 The fields of the structure have the following meanings:

651                   *ulwordSize*      wordsize of RC5 cipher in bytes

652                   *ulRounds*        number of rounds of RC5 encipherment

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

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

## 655 2.5.4 RC5 key generation

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

658 It does not have a parameter.

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

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

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

## 666 2.5.5 RC5-ECB

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

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

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

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

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

683 *Table 15, RC5-ECB Key and Data Length*

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

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

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

## 2.5.6 RC5-CBC

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

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

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

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

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

*Table 16, RC5-CBC Key and Data Length*

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

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

## 2.5.7 RC5-CBC with PKCS padding

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

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

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

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

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

Table 17, RC5-CBC with PKCS Padding; Key and Data Length

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

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

## 2.5.8 General-length RC5-MAC

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

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

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

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

Table 18, General-length RC2-MAC: Key and Data Length

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

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

## 2.5.9 RC5-MAC

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

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

Table 19, RC5-MAC: Key and Data Length

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

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

## 2.6 General block cipher

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

### 2.6.1 Definitions

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

Mechanisms:

- CKM\_DES\_KEY\_GEN
- CKM\_DES\_ECB
- CKM\_DES\_CBC
- CKM\_DES\_MAC
- CKM\_DES\_MAC\_GENERAL
- CKM\_DES\_CBC\_PAD
- CKM\_CDMF\_KEY\_GEN
- CKM\_CDMF\_ECB
- CKM\_CDMF\_CBC
- CKM\_CDMF\_MAC
- CKM\_CDMF\_MAC\_GENERAL
- CKM\_CDMF\_CBC\_PAD
- CKM\_DES\_OFB64
- CKM\_DES\_OFB8
- CKM\_DES\_CFB64
- CKM\_DES\_CFB8
- CKM\_CAST\_KEY\_GEN
- CKM\_CAST\_ECB
- CKM\_CAST\_CBC
- CKM\_CAST\_MAC
- CKM\_CAST\_MAC\_GENERAL
- CKM\_CAST\_CBC\_PAD
- CKM\_CAST3\_KEY\_GEN
- CKM\_CAST3\_ECB
- CKM\_CAST3\_CBC
- CKM\_CAST3\_MAC
- CKM\_CAST3\_MAC\_GENERAL



780 CKM\_CAST3\_CBC\_PAD  
781 CKM\_CAST5\_KEY\_GEN  
782 CKM\_CAST128\_KEY\_GEN  
783 CKM\_CAST5\_ECB  
784 CKM\_CAST128\_ECB  
785 CKM\_CAST5\_CBC  
786 CKM\_CAST128\_CBC  
787 CKM\_CAST5\_MAC  
788 CKM\_CAST128\_MAC  
789 CKM\_CAST5\_MAC\_GENERAL  
790 CKM\_CAST128\_MAC\_GENERAL  
791 CKM\_CAST5\_CBC\_PAD  
792 CKM\_CAST128\_CBC\_PAD  
793 CKM\_IDEA\_KEY\_GEN  
794 CKM\_IDEA\_ECB  
795 CKM\_IDEA\_MAC  
796 CKM\_IDEA\_MAC\_GENERAL  
797 CKM\_IDEA\_CBC\_PAD

## 798 2.6.2 DES secret key objects

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

802 *Table 20, DES Secret Key Object*

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

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

804 DES keys must always have their parity bits properly set as described in FIPS PUB 46-3. Attempting to  
805 create or unwrap a DES key with incorrect parity will return an error.

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

```
807 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
808 CK_KEY_TYPE keyType = CKK_DES;
809 CK_UTF8CHAR label[] = "A DES secret key object";
810 CK_BYTE value[8] = {...};
811 CK_BBOOL true = CK_TRUE;
812 CK_ATTRIBUTE template[] = {
813     {CKA_CLASS, &class, sizeof(class)},
814     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
815     {CKA_TOKEN, &true, sizeof(true)},
816     {CKA_LABEL, label, sizeof(label)-1},
817     {CKA_ENCRYPT, &true, sizeof(true)},
818     {CKA_VALUE, value, sizeof(value)}
819 };
```

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



### 2.6.3 CAST secret key objects

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

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

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

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

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

### 2.6.4 CAST3 secret key objects

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

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

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

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST3;
CK_UTF8CHAR label[] = "A CAST3 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.6.5 CAST128 (CAST5) secret key objects

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

Table 23, CAST128 (CAST5) Secret Key Object Attributes

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

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

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

```
CK OBJECT CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST128;
CK_UTF8CHAR label[] = "A CAST128 secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.6.6 IDEA secret key objects

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

Table 24, IDEA Secret Key Object

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

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

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

```
CK OBJECT CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_IDEA;
CK_UTF8CHAR label[] = "An IDEA secret key object";
CK_BYTE value[16] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.6.7 CDMF secret key objects

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

Table 25, CDMF Secret Key Object

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

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

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

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CDMF;
CK_UTF8CHAR label[] = "A CDMF secret key object";
CK_BYTE value[8] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

## 2.6.8 General block cipher mechanism parameters

### 2.6.8.1 CK\_MAC\_GENERAL\_PARAMS; CK\_MAC\_GENERAL\_PARAMS\_PTR

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

```
typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
```

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

## 2.6.9 General block cipher key generation

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

This mechanism does not have a parameter.

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

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

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

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

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

## 2.6.10 General block cipher ECB

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

It does not have a parameter.

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

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

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

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	

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

## 2.6.11 General block cipher CBC

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

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

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

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

Function	Key type	Input length	Output length	Comments
C_Encrypt	<NAME>	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	

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

## 2.6.12 General block cipher CBC with PKCS padding

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

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

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

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

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

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
--	--	-----------	--------

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

### 2.6.13 General-length general block cipher MAC

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

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

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

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

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

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

### 2.6.14 General block cipher MAC

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

This mechanism has no parameters.

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

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]

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



## 2.7 SKIPJACK

### 2.7.1 Definitions

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

Mechanisms:

- CKM\_SKIPJACK\_KEY\_GEN
- CKM\_SKIPJACK\_ECB64
- CKM\_SKIPJACK\_CBC64
- CKM\_SKIPJACK\_OFB64
- CKM\_SKIPJACK\_CFB64
- CKM\_SKIPJACK\_CFB32
- CKM\_SKIPJACK\_CFB16
- CKM\_SKIPJACK\_CFB8
- CKM\_SKIPJACK\_WRAP
- CKM\_SKIPJACK\_PRIVATE\_WRAP
- CKM\_SKIPJACK\_RELAYX

### 2.7.2 SKIPJACK secret key objects

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

Table 31, SKIPJACK Secret Key Object

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

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

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

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

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SKIPJACK;
CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";
CK_BYTE value[12] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

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

```
1084 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1085 CK_KEY_TYPE keyType = CKK_SKIPJACK;
1086 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
1087 CK_BYTE value[12] = {...};
1088 CK_BBOOL true = CK_TRUE;
1089 CK_ATTRIBUTE template[] = {
1090     {CKA_CLASS, &class, sizeof(class)},
1091     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1092     {CKA_TOKEN, &true, sizeof(true)},
1093     {CKA_LABEL, label, sizeof(label)-1},
1094     {CKA_ENCRYPT, &true, sizeof(true)},
1095     {CKA_WRAP, &true, sizeof(true)},
1096     {CKA_VALUE, value, sizeof(value)}
1097 };
```

1098 **2.7.3 SKIPJACK Mechanism parameters**

1099 **2.7.3.1 CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS;**  
1100 **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR**

1101 **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS** is a structure that provides the parameters to the  
1102 **CKM\_SKIPJACK\_PRIVATE\_WRAP** mechanism. It is defined as follows:

```
1103 typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
1104     CK_ULONG ulPasswordLen;
1105     CK_BYTE_PTR pPassword;
1106     CK_ULONG ulPublicDataLen;
1107     CK_BYTE_PTR pPublicData;
1108     CK_ULONG ulPandGLen;
1109     CK_ULONG ulQLen;
1110     CK_ULONG ulRandomLen;
1111     CK_BYTE_PTR pRandomA;
1112     CK_BYTE_PTR pPrimeP;
1113     CK_BYTE_PTR pBaseG;
1114     CK_BYTE_PTR pSubprimeQ;
1115 } CK_SKIPJACK_PRIVATE_WRAP_PARAMS;
```

1116 The fields of the structure have the following meanings:

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



1127                    *pSubprimeQ*      pointer to Subprime, q, value

1128    **CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR** is a pointer to a  
1129                    **CK\_PRIVATE\_WRAP\_PARAMS**.

1130    **2.7.3.2 CK\_SKIPJACK\_RELAYX\_PARAMS;**  
1131                    **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR**

1132    **CK\_SKIPJACK\_RELAYX\_PARAMS** is a structure that provides the parameters to the  
1133    **CKM\_SKIPJACK\_RELAYX** mechanism. It is defined as follows:

```
1134    typedef struct CK_SKIPJACK_RELAYX_PARAMS {  
1135       CK_ULONG ulOldWrappedXLen;  
1136       CK_BYTE_PTR pOldWrappedX;  
1137       CK_ULONG ulOldPasswordLen;  
1138       CK_BYTE_PTR pOldPassword;  
1139       CK_ULONG ulOldPublicDataLen;  
1140       CK_BYTE_PTR pOldPublicData;  
1141       CK_ULONG ulOldRandomLen;  
1142       CK_BYTE_PTR pOldRandomA;  
1143       CK_ULONG ulNewPasswordLen;  
1144       CK_BYTE_PTR pNewPassword;  
1145       CK_ULONG ulNewPublicDataLen;  
1146       CK_BYTE_PTR pNewPublicData;  
1147       CK_ULONG ulNewRandomLen;  
1148       CK_BYTE_PTR pNewRandomA;  
1149    } CK_SKIPJACK_RELAYX_PARAMS;
```

1150    The fields of the structure have the following meanings:

1151                    *ulOldWrappedLen*      length of old wrapped key in bytes

1152                    *pOldWrappedX*       pointer to old wrapper key

1153                    *ulOldPasswordLen*    length of the old password

1154                    *pOldPassword*       pointer to the buffer which contains the old user-supplied  
1155                    password

1156                    *ulOldPublicDataLen*    old key exchange public key size

1157                    *pOldPublicData*       pointer to old key exchange public key value

1158                    *ulOldRandomLen*       size of old random Ra in bytes

1159                    *pOldRandomA*       pointer to old Ra data

1160                    *ulNewPasswordLen*    length of the new password

1161                    *pNewPassword*       pointer to the buffer which contains the new user-  
1162                    supplied password

1163                    *ulNewPublicDataLen*    new key exchange public key size

1164                    *pNewPublicData*       pointer to new key exchange public key value

1165 *ulNewRandomLen* size of new random Ra in bytes

1166 *pNewRandomA* pointer to new Ra data

1167 **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR** is a pointer to a **CK\_SKIPJACK\_RELAYX\_PARAMS**.

## 1168 2.7.4 SKIPJACK key generation

1169 The SKIPJACK key generation mechanism, denoted **CKM\_SKIPJACK\_KEY\_GEN**, is a key generation  
1170 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1171 It does not have a parameter.

1172 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1173 key.

## 1174 2.7.5 SKIPJACK-ECB64

1175 SKIPJACK-ECB64, denoted **CKM\_SKIPJACK\_ECB64**, is a mechanism for single- and multiple-part  
1176 encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB  
1177 185.

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 can't specify a particular IV when  
1180 encrypting. It can, of course, specify a particular IV when decrypting.

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

1182 *Table 32, SKIPJACK-ECB64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

## 1183 2.7.6 SKIPJACK-CBC64

1184 SKIPJACK-CBC64, denoted **CKM\_SKIPJACK\_CBC64**, is a mechanism for single- and multiple-part  
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 cannot specify a particular IV when  
1188 encrypting. It can, of course, specify a particular IV when decrypting.

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

1190 *Table 33, SKIPJACK-CBC64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

## 1191 2.7.7 SKIPJACK-OFB64

1192 SKIPJACK-OFB64, denoted **CKM\_SKIPJACK\_OFB64**, is a mechanism for single- and multiple-part  
1193 encryption and decryption with SKIPJACK in 64-bit output feedback mode as defined in FIPS PUB 185.

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

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

1198 *Table 34, SKIPJACK-OFB64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

### 1199 2.7.8 SKIPJACK-CFB64

1200 SKIPJACK-CFB64, denoted **CKM\_SKIPJACK\_CFB64**, is a mechanism for single- and multiple-part  
1201 encryption and decryption with SKIPJACK in 64-bit cipher feedback mode as defined in FIPS PUB 185.

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

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

1206 *Table 35, SKIPJACK-CFB64: Data and Length*

Function	Key type	Input length	Output length	Comments
C_Encrypt	SKIPJACK	Multiple of 8	Same as input length	No final part
C_Decrypt	SKIPJACK	Multiple of 8	Same as input length	No final part

### 1207 2.7.9 SKIPJACK-CFB32

1208 SKIPJACK-CFB32, denoted **CKM\_SKIPJACK\_CFB32**, is a mechanism for single- and multiple-part  
1209 encryption and decryption with SKIPJACK in 32-bit cipher feedback mode as defined in FIPS PUB 185.

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

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

1214 *Table 36, SKIPJACK-CFB32: Data and Length*

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

### 1215 2.7.10 SKIPJACK-CFB16

1216 SKIPJACK-CFB16, denoted **CKM\_SKIPJACK\_CFB16**, is a mechanism for single- and multiple-part  
1217 encryption and decryption with SKIPJACK in 16-bit cipher feedback mode as defined in FIPS PUB 185.

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

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

1222 *Table 37, SKIPJACK-CFB16: Data and Length*

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

C_Decrypt	SKIPJACK	Multiple of 4	Same as input length	No final part
-----------	----------	---------------	----------------------	---------------

### 2.7.11 SKIPJACK-CFB8

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

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

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

*Table 38, SKIPJACK-CFB8: Data and Length*

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

### 2.7.12 SKIPJACK-WRAP

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

It does not have a parameter.

### 2.7.13 SKIPJACK-PRIVATE-WRAP

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

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

### 2.7.14 SKIPJACK-RELAYX

The SKIPJACK-RELAYX mechanism, denoted **CKM\_SKIPJACK\_RELAYX**, is used with the **C\_WrapKey** function to “change the wrapping” on a private key which was wrapped with the SKIPJACK-PRIVATE-WRAP mechanism (See Section 2.7.13).

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

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

## 2.8 BATON

### 2.8.1 Definitions

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

Mechanisms:

CKM\_BATON\_KEY\_GEN

CKM\_BATON\_ECB128

CKM\_BATON\_ECB96

1257 CKM\_BATON\_CBC128  
1258 CKM\_BATON\_COUNTER  
1259 CKM\_BATON\_SHUFFLE  
1260 CKM\_BATON\_WRAP

1261 **2.8.2 BATON secret key objects**

1262 BATON secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_BATON**) hold single-length  
1263 BATON keys. The following table defines the BATON secret key object attributes, in addition to the  
1264 common attributes defined for this object class:

1265 *Table 39, BATON Secret Key Object*

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

1266 Refer to [PKCS #11-Base] table 15 for footnotes

1267

1268 BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or  
1269 unwrap a BATON key with incorrect checksum bits will return an error.

1270 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key  
1271 with a specified value. Nonetheless, we provide templates for doing so.

1272 The following is a sample template for creating a BATON MEK secret key object:

```
1273 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1274 CK_KEY_TYPE keyType = CKK_BATON;  
1275 CK_UTF8CHAR label[] = "A BATON MEK secret key object";  
1276 CK_BYTE value[40] = {...};  
1277 CK_BBOOL true = CK_TRUE;  
1278 CK_ATTRIBUTE template[] = {  
1279     {CKA_CLASS, &class, sizeof(class)},  
1280     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1281     {CKA_TOKEN, &true, sizeof(true)},  
1282     {CKA_LABEL, label, sizeof(label)-1},  
1283     {CKA_ENCRYPT, &true, sizeof(true)},  
1284     {CKA_VALUE, value, sizeof(value)}  
1285 };
```

1286 The following is a sample template for creating a BATON TEK secret key object:

```
1287 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1288 CK_KEY_TYPE keyType = CKK_BATON;  
1289 CK_UTF8CHAR label[] = "A BATON TEK secret key object";  
1290 CK_BYTE value[40] = {...};  
1291 CK_BBOOL true = CK_TRUE;  
1292 CK_ATTRIBUTE template[] = {  
1293     {CKA_CLASS, &class, sizeof(class)},  
1294     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1295     {CKA_TOKEN, &true, sizeof(true)},  
1296     {CKA_LABEL, label, sizeof(label)-1},  
1297     {CKA_ENCRYPT, &true, sizeof(true)},  
1298     {CKA_WRAP, &true, sizeof(true)},  
1299     {CKA_VALUE, value, sizeof(value)}  
1300 };
```

1301 **2.8.3 BATON key generation**

1302 The BATON key generation mechanism, denoted **CKM\_BATON\_KEY\_GEN**, is a key generation  
1303 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

It does not have a parameter.

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

## 2.8.4 BATON-ECB128

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

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

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

*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

## 2.8.5 BATON-ECB96

BATON-ECB96, denoted **CKM\_BATON\_ECB96**, is a mechanism for single- and multiple-part encryption and decryption with BATON in 96-bit electronic codebook mode.

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

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

*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

## 2.8.6 BATON-CBC128

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

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

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

*Table 42, BATON-CBC128*

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



## 2.8.7 BATON-COUNTER

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

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

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

*Table 43, BATON-COUNTER: Data and Length*

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

## 2.8.8 BATON-SHUFFLE

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

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

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

*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

## 2.8.9 BATON WRAP

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

It has no parameters.

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

## 2.9 JUNIPER

### 2.9.1 Definitions

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

Mechanisms:

CKM\_JUNIPER\_KEY\_GEN

CKM\_JUNIPER\_ECB128

CKM\_JUNIPER\_CBC128

CKM\_JUNIPER\_COUNTER

CKM\_JUNIPER\_SHUFFLE

1363 CKM\_JUNIPER\_WRAP

1364 **2.9.2 JUNIPER secret key objects**

1365 JUNIPER secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_JUNIPER**) hold single-  
1366 length JUNIPER keys. The following table defines the BATON secret key object attributes, in addition to  
1367 the common attributes defined for this object class:

1368 *Table 45, JUNIPER Secret Key Object*

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

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

1370  
1371 JUNIPER keys have 160 checksum bits, and these bits must be properly set. Attempting to create or  
1372 unwrap a BATON key with incorrect checksum bits will return an error.  
1373 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key  
1374 with a specified value. Nonetheless, we provide templates for doing so.

1375 The following is a sample template for creating a JUNIPER MEK secret key object:

```
1376 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1377 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1378 CK_UTF8CHAR label[] = "A JUNIPER MEK secret key object";  
1379 CK_BYTE value[40] = {...};  
1380 CK_BBOOL true = CK_TRUE;  
1381 CK_ATTRIBUTE template[] = {  
1382     {CKA_CLASS, &class, sizeof(class)},  
1383     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1384     {CKA_TOKEN, &true, sizeof(true)},  
1385     {CKA_LABEL, label, sizeof(label)-1},  
1386     {CKA_ENCRYPT, &true, sizeof(true)},  
1387     {CKA_VALUE, value, sizeof(value)}  
1388 };
```

1389 The following is a sample template for creating a JUNIPER TEK secret key object:

```
1390 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1391 CK_KEY_TYPE keyType = CKK_JUNIPER;  
1392 CK_UTF8CHAR label[] = "A JUNIPER TEK secret key object";  
1393 CK_BYTE value[40] = {...};  
1394 CK_BBOOL true = CK_TRUE;  
1395 CK_ATTRIBUTE template[] = {  
1396     {CKA_CLASS, &class, sizeof(class)},  
1397     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1398     {CKA_TOKEN, &true, sizeof(true)},  
1399     {CKA_LABEL, label, sizeof(label)-1},  
1400     {CKA_ENCRYPT, &true, sizeof(true)},  
1401     {CKA_WRAP, &true, sizeof(true)},  
1402     {CKA_VALUE, value, sizeof(value)}  
1403 };
```

1404 **2.9.3 JUNIPER key generation**

1405 The JUNIPER key generation mechanism, denoted **CKM\_JUNIPER\_KEY\_GEN**, is a key generation  
1406 mechanism for JUNIPER. The output of this mechanism is called a Message Encryption Key (MEK).  
1407 It does not have a parameter.

1408 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1409 key.



## 2.9.4 JUNIPER-ECB128

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

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

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

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

## 2.9.5 JUNIPER-CBC128

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

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

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

Table 47, JUNIPER-CBC128: Data and Length

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

## 2.9.6 JUNIPER-COUNTER

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

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

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

Table 48, JUNIPER-COUNTER: Data and Length

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

## 2.9.7 JUNIPER-SHUFFLE

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

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

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

*Table 49, JUNIPER-SHUFFLE: Data and Length*

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

## 2.9.8 JUNIPER WRAP

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

It has no parameters.

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

## 2.10 MD2

### 2.10.1 Definitions

Mechanisms:

- CKM\_MD2
- CKM\_MD2\_HMAC
- CKM\_MD2\_HMAC\_GENERAL
- CKM\_MD2\_KEY\_DERIVATION

### 2.10.2 MD2 digest

The MD2 mechanism, denoted **CKM\_MD2**, is a mechanism for message digesting, following the MD2 message-digest algorithm defined in RFC 1319.

It does not have a parameter.

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

*Table 50, MD2: Data Length*

Function	Data length	Digest Length
C_Digest	Any	16

### 2.10.3 General-length MD2-HMAC

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

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

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

## 2.10.4 MD2-HMAC

The MD2-HMAC mechanism, denoted **CKM\_MD2\_HMAC**, is a special case of the general-length MD2-HMAC mechanism in Section 2.10.3.

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

## 2.10.5 MD2 key derivation

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

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

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

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

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

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

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

## 2.11 MD5

### 2.11.1 Definitions

Mechanisms:

CKM\_MD5

CKM\_MD5\_HMAC

1511 CKM\_MD5\_HMAC\_GENERAL  
1512 CKM\_MD5\_KEY\_DERIVATION

1513 **2.11.2 MD5 Digest**

1514 The MD5 mechanism, denoted **CKM\_MD5**, is a mechanism for message digesting, following the MD5  
1515 message-digest algorithm defined in RFC 1321.

1516 It does not have a parameter.

1517 Constraints on the length of input and output data are summarized in the following table. For single-part  
1518 digesting, the data and the digest may begin at the same location in memory.

1519 *Table 52, MD5: Data Length*

Function	Data length	Digest length
C_Digest	Any	16

1520 **2.11.3 General-length MD5-HMAC**

1521 The general-length MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC\_GENERAL**, is a mechanism for  
1522 signatures and verification. It uses the HMAC construction, based on the MD5 hash function. The keys it  
1523 uses are generic secret keys.

1524 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1525 output. This length should be in the range 0-16 (the output size of MD5 is 16 bytes). Signatures (MACs)  
1526 produced by this mechanism will be taken from the start of the full 16-byte HMAC output.

1527 *Table 53, General-length MD5-HMAC: Key and Data Length*

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

1528 **2.11.4 MD5-HMAC**

1529 The MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC**, is a special case of the general-length MD5-  
1530 HMAC mechanism in Section 2.11.3.

1531 It has no parameter, and always produces an output of length 16.

1532 **2.11.5 MD5 key derivation**

1533 MD5 key derivation denoted **CKM\_MD5\_KEY\_DERIVATION**, is a mechanism which provides the  
1534 capability of deriving a secret key by digesting the value of another secret key with MD5.

1535 The value of the base key is digested once, and the result is used to make the value of derived secret  
1536 key.

- 1537 • If no length or key type is provided in the template, then the key produced by this mechanism will be a  
1538 generic secret key. Its length will be 16 bytes (the output size of MD5).
- 1539 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
1540 will be a generic secret key of the specified length.
- 1541 • If no length was provided in the template, but a key type is, then that key type must have a well-  
1542 defined length. If it does, then the key produced by this mechanism will be of the type specified in the  
1543 template. If it doesn't, an error will be returned.
- 1544 • If both a key type and a length are provided in the template, the length must be compatible with that  
1545 key type. The key produced by this mechanism will be of the specified type and length.

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

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

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

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both be specified to either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

## 2.12 FASTHASH

### 2.12.1 Definitions

Mechanisms:  
CKM\_FASTHASH

### 2.12.2 FASTHASH digest

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

It does not have a parameter.

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

*Table 54, FASTHASH: Data Length*

Function	Input length	Digest length
C_Digest	Any	40

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

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

### 2.13.1 Definitions

Mechanisms:

- CKM\_PBE\_MD2\_DES\_CBC
- CKM\_PBE\_MD5\_DES\_CBC
- CKM\_PBE\_MD5\_CAST\_CBC
- CKM\_PBE\_MD5\_CAST3\_CBC
- CKM\_PBE\_MD5\_CAST5\_CBC
- CKM\_PBE\_MD5\_CAST128\_CBC
- CKM\_PBE\_SHA1\_CAST5\_CBC
- CKM\_PBE\_SHA1\_CAST128\_CBC

1584 CKM\_PBE\_SHA1\_RC4\_128  
1585 CKM\_PBE\_SHA1\_RC4\_40  
1586 CKM\_PBE\_SHA1\_RC2\_128\_CBC  
1587 CKM\_PBE\_SHA1\_RC2\_40\_CBC

## 1588 2.13.2 Password-based encryption/authentication mechanism parameters

### 1589 2.13.2.1 CK\_PBE\_PARAMS; CK\_PBE\_PARAMS\_PTR

1590 **CK\_PBE\_PARAMS** is a structure which provides all of the necessary information required by the  
1591 CKM\_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation  
1592 mechanisms) and the CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC mechanism. It is defined as follows:

```
1593 typedef struct CK_PBE_PARAMS {  
1594     CK_BYTE_PTR pInitVector;  
1595     CK_UTF8CHAR_PTR pPassword;  
1596     CK_ULONG ulPasswordLen;  
1597     CK_BYTE_PTR pSalt;  
1598     CK_ULONG ulSaltLen;  
1599     CK_ULONG ulIteration;  
1600 } CK_PBE_PARAMS;
```

1601 The fields of the structure have the following meanings:

1602	<i>pInitVector</i>	pointer to the location that receives the 8-byte
1603		initialization vector (IV), if an IV is required
1604	<i>pPassword</i>	points to the password to be used in the PBE key
1605		generation
1606	<i>ulPasswordLen</i>	length in bytes of the password information
1607	<i>pSalt</i>	points to the salt to be used in the PBE key generation
1608	<i>ulSaltLen</i>	length in bytes of the salt information
1609	<i>ulIteration</i>	number of iterations required for the generation

1610 **CK\_PBE\_PARAMS\_PTR** is a pointer to a **CK\_PBE\_PARAMS**.

### 1611 2.13.3 MD2-PBE for DES-CBC

1612 MD2-PBE for DES-CBC, denoted **CKM\_PBE\_MD2\_DES\_CBC**, is a mechanism used for generating a  
1613 DES secret key and an IV from a password and a salt value by using the MD2 digest algorithm and an  
1614 iteration count. This functionality is defined in PKCS #5 as PBKDF1.

1615 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1616 key generation process and the location of the application-supplied buffer which will receive the 8-byte IV  
1617 generated by the mechanism.

### 1618 2.13.4 MD5-PBE for DES-CBC

1619 MD5-PBE for DES-CBC, denoted **CKM\_PBE\_MD5\_DES\_CBC**, is a mechanism used for generating a  
1620 DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an  
1621 iteration count. This functionality is defined in PKCS #5 as PBKDF1.



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

### 2.13.5 MD5-PBE for CAST-CBC

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

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

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

### 2.13.6 MD5-PBE for CAST3-CBC

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

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

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

### 2.13.7 MD5-PBE for CAST128-CBC (CAST5-CBC)

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

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

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

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

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

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

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

## 2.14 PKCS #12 password-based encryption/authentication mechanisms

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

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

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

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

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

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

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

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

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

It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the key generation process. The parameter also has a field to hold the location of an application-supplied buffer which will receive an IV; for this mechanism, the contents of this field are ignored, since RC4 does not require an IV.



1710 The key produced by this mechanism will typically be used for performing password-based encryption.

## 1711 2.14.2 SHA-1\_PBE for 40-bit RC4

1712 SHA-1-PBE for 40-bit RC4, denoted **CKM\_PBE\_SHA1\_RC4\_40**, is a mechanism used for generating a  
1713 40-bit RC4 secret key from a password and a salt value by using the SHA-1 digest algorithm and an  
1714 iteration count. The method used to generate the key is described above.

1715 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1716 key generation process. The parameter also has a field to hold the location of an application-supplied  
1717 buffer which will receive an IV; for this mechanism, the contents of this field are ignored, since RC4 does  
1718 not require an IV.

1719 The key produced by this mechanism will typically be used for performing password-based encryption.

## 1720 2.14.3 SHA-1\_PBE for 128-bit RC2-CBC

1721 SHA-1-PBE for 128-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_128\_CBC**, is a mechanism used for  
1722 generating a 128-bit RC2 secret key from a password and a salt value by using the SHA-1 digest  
1723 algorithm and an iteration count. The method used to generate the key and IV is described above.

1724 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1725 key generation process and the location of an application-supplied buffer which will receive the 8-byte IV  
1726 generated by the mechanism.

1727 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number  
1728 of bits in the RC2 search space should be set to 128. This ensures compatibility with the ASN.1 Object  
1729 Identifier `pbeWithSHA1And128BitRC2-CBC`.

1730 The key and IV produced by this mechanism will typically be used for performing password-based  
1731 encryption.

## 1732 2.14.4 SHA-1\_PBE for 40-bit RC2-CBC

1733 SHA-1-PBE for 40-bit RC2-CBC, denoted **CKM\_PBE\_SHA1\_RC2\_40\_CBC**, is a mechanism used for  
1734 generating a 40-bit RC2 secret key from a password and a salt value by using the SHA-1 digest algorithm  
1735 and an iteration count. The method used to generate the key and IV is described above.

1736 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
1737 key generation process and the location of an application-supplied buffer which will receive the 8-byte IV  
1738 generated by the mechanism.

1739 When the key and IV generated by this mechanism are used to encrypt or decrypt, the effective number  
1740 of bits in the RC2 search space should be set to 40. This ensures compatibility with the ASN.1 Object  
1741 Identifier `pbeWithSHA1And40BitRC2-CBC`.

1742 The key and IV produced by this mechanism will typically be used for performing password-based  
1743 encryption

## 1744 2.15 RIPE-MD

### 1745 2.15.1 Definitions

1746 Mechanisms:

1747 CKM\_RIPEMD128

1748 CKM\_RIPEMD128\_HMAC

1749 CKM\_RIPEMD128\_HMAC\_GENERAL

1750 CKM\_RIPEMD160

1751 CKM\_RIPEMD160\_HMAC

1752 CKM\_RIPEMD160\_HMAC\_GENERAL

## 2.15.2 RIPE-MD 128 Digest

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

It does not have a parameter.

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

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

Function	Data length	Digest length
C_Digest	Any	16

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

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

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

*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

## 2.15.4 RIPE-MD 128-HMAC

The RIPE-MD 128-HMAC mechanism, denoted **CKM\_RIPEMD128\_HMAC**, is a special case of the general-length RIPE-MD 128-HMAC mechanism in Section 2.15.3.

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

## 2.15.5 RIPE-MD 160

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

It does not have a parameter.

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

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

Function	Data length	Digest length
C_Digest	Any	20

## 2.15.6 General-length RIPE-MD 160-HMAC

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

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

1785 *Table 58, General-length RIPE-MD 160-HMAC: Data and Length*

Function	Key type	Data length	Signature length
C_Sign	Generic secret	Any	0-20, depending on parameters
C_Verify	Generic secret	Any	0-20, depending on parameters

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

1787 The RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC**, is a special case of the  
1788 general-length RIPE-MD 160HMAC mechanism in Section 2.15.6.

1789 It has no parameter, and always produces an output of length 20.

1790 **2.16 SET**

1791 **2.16.1 Definitions**

1792 Mechanisms:

1793 **CKM\_KEY\_WRAP\_SET\_OAEP**

1794 **2.16.2 SET mechanism parameters**

1795 **2.16.2.1 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS;**  
1796 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR**

1797 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** is a structure that provides the parameters to the  
1798 **CKM\_KEY\_WRAP\_SET\_OAEP** mechanism. It is defined as follows:

```
1799 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {  
1800     CK_BYTE bBC;  
1801     CK_BYTE_PTR pX;  
1802     CK_ULONG ulXLen;  
1803 } CK_KEY_WRAP_SET_OAEP_PARAMS;
```

1804 The fields of the structure have the following meanings:

1805 *bBC* block contents byte

1806 *pX* concatenation of hash of plaintext data (if present) and  
1807 extra data (if present)

1808 *ulXLen* length in bytes of concatenation of hash of plaintext data  
1809 (if present) and extra data (if present). 0 if neither is  
1810 present.

1811 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR** is a pointer to a  
1812 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS**.

1813 **2.16.3 OAEP key wrapping for SET**

1814 The OAEP key wrapping for SET mechanism, denoted **CKM\_KEY\_WRAP\_SET\_OAEP**, is a mechanism  
1815 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some

extra data may optionally be wrapped together with the DES key. This mechanism is defined in the SET protocol specifications.

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

When this mechanism is used to unwrap a key, the concatenation of the hash of plaintext data (if present) and the extra data (if present) is returned following the convention described in Section \*\*\*MISSING REFERENCE\*\*\* on producing output. Note that if the inputs to **C\_UnwrapKey** are such that the extra data is not returned (e.g. the buffer supplied in the **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure is **NULL\_PTR**), then the unwrapped key object will not be created, either.

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

If an application uses **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP**, it may be preferable for it simply to allocate a 128-byte buffer for the concatenation of the hash of plaintext data and the extra data (this concatenation is never larger than 128 bytes), rather than calling **C\_UnwrapKey** twice. Each call of **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP** requires an RSA decryption operation to be performed, and this computational overhead can be avoided by this means.

## 2.17 LYNKS

### 2.17.1 Definitions

Mechanisms:

**CKM\_KEY\_WRAP\_LYNKS**

### 2.17.2 LYNKS key wrapping

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

It does not have a parameter.

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

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

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

---

### 3 PKCS #11 Implementation Conformance

1857

1858 An implementation is a conforming implementation if it meets the conditions specified in one or more  
1859 server profiles specified in **[PKCS #11-Prof]**.

1860 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1861 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL  
1862 conform to all normative statements within the clauses specified for that profile and for any subclauses to  
1863 each of those clauses .

1864

---

## Appendix A. Acknowledgments

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

### Participants:

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

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

---

## Appendix B. Manifest constants

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

```
/*
 * Copyright OASIS Open 2013. All rights reserved.
 * OASIS trademark, IPR and other policies apply.
 * http://www.oasis-open.org/policies-guidelines/ipr
 */

#define CKK_KEA 0x00000005
#define CKK_RC2 0x00000011
#define CKK_RC4 0x00000012
#define CKK_DES 0x00000013
#define CKK_CAST 0x00000016
#define CKK_CAST3 0x00000017
#define CKK_CAST5 0x00000018
#define CKK_CAST128 0x00000018
#define CKK_RC5 0x00000019
#define CKK_IDEA 0x0000001A
#define CKK_SKIPJACK 0x0000001B
#define CKK_BATON 0x0000001C
#define CKK_JUNIPER 0x0000001D
#define CKM_MD2_RSA_PKCS 0x00000004
#define CKM_MD5_RSA_PKCS 0x00000005
#define CKM_RIPEMD128_RSA_PKCS 0x00000007
#define CKM_RIPEMD160_RSA_PKCS 0x00000008
#define CKM_RC2_KEY_GEN 0x00000100
#define CKM_RC2_ECB 0x00000101
#define CKM_RC2_CBC 0x00000102
#define CKM_RC2_MAC 0x00000103
#define CKM_RC2_MAC_GENERAL 0x00000104
#define CKM_RC2_CBC_PAD 0x00000105
#define CKM_RC4_KEY_GEN 0x00000110
#define CKM_RC4 0x00000111
#define CKM_DES_KEY_GEN 0x00000120
#define CKM_DES_ECB 0x00000121
#define CKM_DES_CBC 0x00000122
#define CKM_DES_MAC 0x00000123
#define CKM_DES_MAC_GENERAL 0x00000124
#define CKM_DES_CBC_PAD 0x00000125
#define CKM_MD2 0x00000200
#define CKM_MD2_HMAC 0x00000201
#define CKM_MD2_HMAC_GENERAL 0x00000202
#define CKM_MD5 0x00000210
#define CKM_MD5_HMAC 0x00000211
#define CKM_MD5_HMAC_GENERAL 0x00000212
#define CKM_RIPEMD128 0x00000230
#define CKM_RIPEMD128_HMAC 0x00000231
#define CKM_RIPEMD128_HMAC_GENERAL 0x00000232
#define CKM_RIPEMD160 0x00000240
#define CKM_RIPEMD160_HMAC 0x00000241
#define CKM_RIPEMD160_HMAC_GENERAL 0x00000242
#define CKM_CAST_KEY_GEN 0x00000300
#define CKM_CAST_ECB 0x00000301
#define CKM_CAST_CBC 0x00000302
#define CKM_CAST_MAC 0x00000303
#define CKM_CAST_MAC_GENERAL 0x00000304
#define CKM_CAST_CBC_PAD 0x00000305
#define CKM_CAST3_KEY_GEN 0x00000310
```



```

1994 #define CKM_CAST3_ECB 0x00000311
1995 #define CKM_CAST3_CBC 0x00000312
1996 #define CKM_CAST3_MAC 0x00000313
1997 #define CKM_CAST3_MAC_GENERAL 0x00000314
1998 #define CKM_CAST3_CBC_PAD 0x00000315
1999 #define CKM_CAST5_KEY_GEN 0x00000320
2000 #define CKM_CAST128_KEY_GEN 0x00000320
2001 #define CKM_CAST5_ECB 0x00000321
2002 #define CKM_CAST128_ECB 0x00000321
2003 #define CKM_CAST5_CBC 0x00000322
2004 #define CKM_CAST128_CBC 0x00000322
2005 #define CKM_CAST5_MAC 0x00000323
2006 #define CKM_CAST128_MAC 0x00000323
2007 #define CKM_CAST5_MAC_GENERAL 0x00000324
2008 #define CKM_CAST128_MAC_GENERAL 0x00000324
2009 #define CKM_CAST5_CBC_PAD 0x00000325
2010 #define CKM_CAST128_CBC_PAD 0x00000325
2011 #define CKM_RC5_KEY_GEN 0x00000330
2012 #define CKM_RC5_ECB 0x00000331
2013 #define CKM_RC5_CBC 0x00000332
2014 #define CKM_RC5_MAC 0x00000333
2015 #define CKM_RC5_MAC_GENERAL 0x00000334
2016 #define CKM_RC5_CBC_PAD 0x00000335
2017 #define CKM_IDEA_KEY_GEN 0x00000340
2018 #define CKM_IDEA_ECB 0x00000341
2019 #define CKM_IDEA_CBC 0x00000342
2020 #define CKM_IDEA_MAC 0x00000343
2021 #define CKM_IDEA_MAC_GENERAL 0x00000344
2022 #define CKM_IDEA_CBC_PAD 0x00000345
2023 #define CKM_MD5_KEY_DERIVATION 0x00000390
2024 #define CKM_MD2_KEY_DERIVATION 0x00000391
2025 #define CKM_PBE_MD2_DES_CBC 0x000003A0
2026 #define CKM_PBE_MD5_DES_CBC 0x000003A1
2027 #define CKM_PBE_MD5_CAST_CBC 0x000003A2
2028 #define CKM_PBE_MD5_CAST3_CBC 0x000003A3
2029 #define CKM_PBE_MD5_CAST5_CBC 0x000003A4
2030 #define CKM_PBE_MD5_CAST128_CBC 0x000003A4
2031 #define CKM_PBE_SHA1_CAST5_CBC 0x000003A5
2032 #define CKM_PBE_SHA1_CAST128_CBC 0x000003A5
2033 #define CKM_PBE_SHA1_RC4_128 0x000003A6
2034 #define CKM_PBE_SHA1_RC4_40 0x000003A7
2035 #define CKM_PBE_SHA1_RC2_128_CBC 0x000003AA
2036 #define CKM_PBE_SHA1_RC2_40_CBC 0x000003AB
2037 #define CKM_KEY_WRAP_LYNKS 0x00000400
2038 #define CKM_KEY_WRAP_SET_OAEP 0x00000401
2039 #define CKM_SKIPJACK_KEY_GEN 0x00001000
2040 #define CKM_SKIPJACK_ECB64 0x00001001
2041 #define CKM_SKIPJACK_CBC64 0x00001002
2042 #define CKM_SKIPJACK_OFB64 0x00001003
2043 #define CKM_SKIPJACK_CFB64 0x00001004
2044 #define CKM_SKIPJACK_CFB32 0x00001005
2045 #define CKM_SKIPJACK_CFB16 0x00001006
2046 #define CKM_SKIPJACK_CFB8 0x00001007
2047 #define CKM_SKIPJACK_WRAP 0x00001008
2048 #define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009
2049 #define CKM_SKIPJACK_RELAYX 0x0000100a
2050 #define CKM_KEA_KEY_PAIR_GEN 0x00001010
2051 #define CKM_KEA_KEY_DERIVE 0x00001011
2052 #define CKM_FORTEZZA_TIMESTAMP 0x00001020
2053 #define CKM_BATON_KEY_GEN 0x00001030
2054 #define CKM_BATON_ECB128 0x00001031
2055 #define CKM_BATON_ECB96 0x00001032
2056 #define CKM_BATON_CBC128 0x00001033
2057 #define CKM_BATON_COUNTER 0x00001034

```

```
2058 #define CKM_BATON_SHUFFLE 0x00001035
2059 #define CKM_BATON_WRAP 0x00001036
2060 #define CKM_JUNIPER_KEY_GEN 0x00001060
2061 #define CKM_JUNIPER_ECB128 0x00001061
2062 #define CKM_JUNIPER_CBC128 0x00001062
2063 #define CKM_JUNIPER_COUNTER 0x00001063
2064 #define CKM_JUNIPER_SHUFFLE 0x00001064
2065 #define CKM_JUNIPER_WRAP 0x00001065
2066 #define CKM_FASTHASH 0x00001070
```

2067

---

## Appendix C. Revision History

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