



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

## Committee Specification 01

19 December 2019

### This stage:

<https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/cs01/pkcs11-hist-v3.0-cs01.docx> (Authoritative)  
<https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/cs01/pkcs11-hist-v3.0-cs01.html>  
<https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/cs01/pkcs11-hist-v3.0-cs01.pdf>

### Previous stage:

N/A

### Latest stage:

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

### Technical Committee:

OASIS PKCS 11 TC

### Chairs:

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

### Editors:

Chris Zimman ([chris@wmpp.com](mailto:chris@wmpp.com)), Individual  
Dieter Bong ([dieter.bong@utimaco.com](mailto:dieter.bong@utimaco.com)), Utimaco IS GmbH

### Additional artifacts:

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

- PKCS #11 header files:  
<https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/cs01/include/pkcs11-v3.0/>

### Related work:

This specification replaces or supersedes:

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

This specification is related to:

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

**Abstract:**

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

**Status:**

This document was last revised or approved by the OASIS PKCS 11 TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=pkcs11#technical](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=pkcs11#technical).

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

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

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

**Citation format:**

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

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

*PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 3.0*. Edited by Chris Zimman and Dieter Bong. 19 December 2019. OASIS Committee Specification 01. <https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/cs01/pkcs11-hist-v3.0-cs01.html>. Latest stage: <https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/pkcs11-hist-v3.0.html>.

---

## Notices

Copyright © OASIS Open 2019. All Rights Reserved.

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

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

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

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

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

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

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

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

---

# Table of Contents

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

|   |    |
|---|----|
| 2.6.5 RC5-ECB.....  | 27 |
| 2.6.6 RC5-CBC.....  | 28 |
| 2.6.7 RC5-CBC with PKCS padding .....   | 28 |
| 2.6.8 General-length RC5-MAC .....  | 29 |
| 2.6.9 RC5-MAC .....   | 29 |
| 2.7 General block cipher .....  | 30 |
| 2.7.1 Definitions.....  | 30 |
| 2.7.2 DES secret key objects .....  | 31 |
| 2.7.3 CAST secret key objects .....   | 31 |
| 2.7.4 CAST3 secret key objects .....  | 32 |
| 2.7.5 CAST128 secret key objects .....  | 32 |
| 2.7.6 IDEA secret key objects .....   | 33 |
| 2.7.7 CDMF secret key objects .....   | 33 |
| 2.7.8 General block cipher mechanism parameters.....                              | 34 |
| 2.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR.....                     | 34 |
| 2.7.9 General block cipher key generation.....                                    | 34 |
| 2.7.10 General block cipher ECB.....  | 35 |
| 2.7.11 General block cipher CBC.....  | 35 |
| 2.7.12 General block cipher CBC with PCKS padding.....                            | 36 |
| 2.7.13 General-length general block cipher MAC .....                              | 36 |
| 2.7.14 General block cipher MAC .....   | 37 |
| 2.8 SKIPJACK.....   | 37 |
| 2.8.1 Definitions.....  | 37 |
| 2.8.2 SKIPJACK secret key objects .....   | 38 |
| 2.8.3 SKIPJACK Mechanism parameters .....   | 39 |
| 2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR..... | 39 |
| 2.8.3.2 CK_SKIPJACK_RELAYX_PARAMS; CK_SKIPJACK_RELAYX_PARAMS_PTR.....             | 39 |
| 2.8.4 SKIPJACK key generation .....   | 40 |
| 2.8.5 SKIPJACK-ECB64 .....  | 41 |
| 2.8.6 SKIPJACK-CBC64 .....  | 41 |
| 2.8.7 SKIPJACK-OFB64 .....  | 41 |
| 2.8.8 SKIPJACK-CFB64.....   | 41 |
| 2.8.9 SKIPJACK-CFB32.....   | 42 |
| 2.8.10 SKIPJACK-CFB16.....  | 42 |
| 2.8.11 SKIPJACK-CFB8.....   | 42 |
| 2.8.12 SKIPJACK-WRAP .....  | 43 |
| 2.8.13 SKIPJACK-PRIVATE-WRAP .....  | 43 |
| 2.8.14 SKIPJACK-RELAYX.....   | 43 |
| 2.9 BATON.....  | 43 |
| 2.9.1 Definitions.....  | 43 |
| 2.9.2 BATON secret key objects .....  | 43 |
| 2.9.3 BATON key generation .....  | 44 |
| 2.9.4 BATON-ECB128 .....  | 44 |
| 2.9.5 BATON-ECB96.....  | 45 |
| 2.9.6 BATON-CBC128 .....  | 45 |
| 2.9.7 BATON-COUNTER .....   | 45 |

|   |    |
|---|----|
| 2.9.8 BATON-SHUFFLE .....   | 46 |
| 2.9.9 BATON WRAP .....  | 46 |
| 2.10 JUNIPER.....   | 46 |
| 2.10.1 Definitions.....   | 46 |
| 2.10.2 JUNIPER secret key objects .....                                   | 46 |
| 2.10.3 JUNIPER key generation .....                                       | 47 |
| 2.10.4 JUNIPER-ECB128 .....   | 47 |
| 2.10.5 JUNIPER-CBC128 .....   | 48 |
| 2.10.6 JUNIPER-COUNTER .....  | 48 |
| 2.10.7 JUNIPER-SHUFFLE .....  | 48 |
| 2.10.8 JUNIPER WRAP .....   | 49 |
| 2.11 MD2 .....  | 49 |
| 2.11.1 Definitions.....   | 49 |
| 2.11.2 MD2 digest .....   | 49 |
| 2.11.3 General-length MD2-HMAC .....                                      | 49 |
| 2.11.4 MD2-HMAC .....   | 50 |
| 2.11.5 MD2 key derivation.....  | 50 |
| 2.12 MD5 .....  | 50 |
| 2.12.1 Definitions.....   | 50 |
| 2.12.2 MD5 Digest.....  | 51 |
| 2.12.3 General-length MD5-HMAC .....                                      | 51 |
| 2.12.4 MD5-HMAC .....   | 51 |
| 2.12.5 MD5 key derivation.....  | 51 |
| 2.13 FASTHASH.....  | 52 |
| 2.13.1 Definitions.....   | 52 |
| 2.13.2 FASTHASH digest.....   | 52 |
| 2.14 PKCS #5 and PKCS #5-style password-based encryption (PBD) .....      | 52 |
| 2.14.1 Definitions.....   | 52 |
| 2.14.2 Password-based encryption/authentication mechanism parameters..... | 53 |
| 2.14.2.1 CK_PBE_PARAMS; CK_PBE_PARAMS_PTR .....                           | 53 |
| 2.14.3 MD2-PBE for DES-CBC .....  | 53 |
| 2.14.4 MD5-PBE for DES-CBC .....  | 53 |
| 2.14.5 MD5-PBE for CAST-CBC.....  | 54 |
| 2.14.6 MD5-PBE for CAST3-CBC.....   | 54 |
| 2.14.7 MD5-PBE for CAST128-CBC.....                                       | 54 |
| 2.14.8 SHA-1-PBE for CAST128-CBC.....                                     | 54 |
| 2.15 PKCS #12 password-based encryption/authentication mechanisms .....   | 54 |
| 2.15.1 Definitions.....   | 54 |
| 2.15.2 SHA-1-PBE for 128-bit RC4 .....                                    | 55 |
| 2.15.3 SHA-1_PBE for 40-bit RC4 .....                                     | 56 |
| 2.15.4 SHA-1_PBE for 128-bit RC2-CBC .....                                | 56 |
| 2.15.5 SHA-1_PBE for 40-bit RC2-CBC .....                                 | 56 |
| 2.16 RIPE-MD.....   | 56 |
| 2.16.1 Definitions.....   | 56 |
| 2.16.2 RIPE-MD 128 Digest .....   | 57 |
| 2.16.3 General-length RIPE-MD 128-HMAC .....                              | 57 |

|   |    |
|---|----|
| 2.16.4 RIPE-MD 128-HMAC .....   | 57 |
| 2.16.5 RIPE-MD 160 .....  | 57 |
| 2.16.6 General-length RIPE-MD 160-HMAC .....                                | 57 |
| 2.16.7 RIPE-MD 160-HMAC .....   | 58 |
| 2.17 SET .....  | 58 |
| 2.17.1 Definitions .....  | 58 |
| 2.17.2 SET mechanism parameters .....                                       | 58 |
| 2.17.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS; CK_KEY_WRAP_SET_OAEP_PARAMS_PTR ..... | 58 |
| 2.17.3 OAEP key wrapping for SET .....                                      | 58 |
| 2.18 LYNKS .....  | 59 |
| 2.18.1 Definitions .....  | 59 |
| 2.18.2 LYNKS key wrapping .....   | 59 |
| 3 PKCS #11 Implementation Conformance .....                                 | 60 |
| Appendix A. Acknowledgments .....   | 61 |
| Appendix B. Manifest constants .....  | 63 |
| Appendix C. Revision History .....  | 64 |

---

# 1 Introduction

## 1.1 IPR Policy

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

## 1.2 Description of this Document

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

All text is normative unless otherwise labeled.

## 1.3 Terminology

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

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



|    |                 |  |
|----|-----------------|--|
| 38 | <b>MD5</b>      | RSA Security's MD5 message-digest algorithm, as defined in RFC     |
| 39 |                 | 1321.  |
| 40 | <b>PRF</b>      | Pseudo random function.  |
| 41 | <b>RSA</b>      | The RSA public-key cryptosystem.                                   |
| 42 | <b>RC2</b>      | RSA Security's RC2 symmetric block cipher.                         |
| 43 | <b>RC4</b>      | RSA Security's proprietary RC4 symmetric stream cipher.            |
| 44 | <b>RC5</b>      | RSA Security's RC5 symmetric block cipher.                         |
| 45 | <b>SET</b>      | <b>The Secure Electronic Transaction protocol.</b>                 |
| 46 | <b>SHA-1</b>    | The (revised) Secure Hash Algorithm with a 160-bit message digest, |
| 47 |                 | as defined in FIPS PUB 180-2.                                      |
| 48 | <b>SKIPJACK</b> | MISSI's SKIPJACK block cipher.                                     |
| 49 |                 |  |

## 50 1.5 Normative References

|    |                        |   |
|----|------------------------|---|
| 51 | <b>[PKCS #11-Base]</b> | <i>PKCS #11 Cryptographic Token Interface Base Specification Version 2.40.</i>  |
| 52 |                        | Edited by Susan Gleeson and Chris Zimman. Latest version. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html">http://docs.oasis-</a> |
| 53 |                        | <a href="http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html">open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html</a> .                   |
| 54 |                        |   |
| 55 | <b>[PKCS #11-Curr]</b> | <i>PKCS #11 Cryptographic Token Interface Current Mechanisms Specification</i>  |
| 56 |                        | <i>Version 2.40.</i> Edited by Susan Gleeson and Chris Zimman. Latest version.  |
| 57 |                        | <a href="http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.html</a> . |
| 58 |                        |   |
| 59 | <b>[PKCS #11-Prof]</b> | <i>PKCS #11 Cryptographic Token Interface Profiles Version 2.40.</i> Edited by Tim  |
| 60 |                        | Hudson. Latest version. <a href="http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html">http://docs.oasis-open.org/pkcs11/pkcs11-</a>    |
| 61 |                        | <a href="http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html">profiles/v2.40/pkcs11-profiles-v2.40.html</a> .                          |
| 62 |                        |   |
| 63 | <b>[RFC2119]</b>       | Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP  |
| 64 |                        | 14, RFC 2119, March 1997. <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> .   |
| 65 |                        |   |

## 66 1.6 Non-Normative References

|    |                        |   |
|----|------------------------|---|
| 67 | <b>[ANSI C]</b>        | ANSI/ISO. American National Standard for Programming Languages – C. 1990  |
| 68 | <b>[ANSI X9.31]</b>    | Accredited Standards Committee X9. Digital Signatures Using Reversible Public   |
| 69 |                        | Key Cryptography for the Financial Services Industry (rDSA). 1998.  |
| 70 | <b>[ANSI X9.42]</b>    | Accredited Standards Committee X9. Public Key Cryptography for the Financial  |
| 71 |                        | Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm   |
| 72 |                        | Cryptography. 2003  |
| 73 | <b>[ANSI X9.62]</b>    | Accredited Standards Committee X9. Public Key Cryptography for the Financial  |
| 74 |                        | Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998   |
| 75 | <b>[CC/PP]</b>         | G. Klyne, F. Reynolds, C. , H. Ohto, J. Hjelm, M. H. Butler, L. Tran, Editors,  |
| 76 |                        | W3C. <i>Composite Capability/Preference Profiles (CC/PP): Structure and</i>   |
| 77 |                        | <i>Vocabularies</i> . 2004, URL: <a href="http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/">http://www.w3.org/TR/2004/REC-CCPP-struct-</a> |
| 78 |                        | <a href="http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/">vocab-20040115/</a>   |
| 79 | <b>[CDPD]</b>          | Ameritech Mobile Communications et al. Cellular Digital Packet Data System  |
| 80 |                        | Specifications: Part 406: Airlink Security. 1993  |
| 81 | <b>[FIPS PUB 46-3]</b> | NIST. <i>FIPS 46-3: Data Encryption Standard (DES)</i> . October 26, 1999. URL:   |
| 82 |                        | <a href="http://csrc.nist.gov/publications/fips/index.html">http://csrc.nist.gov/publications/fips/index.html</a>                                   |

- [FIPS PUB 81]** NIST. *FIPS 81: DES Modes of Operation*. December 1980. URL: <http://csrc.nist.gov/publications/fips/index.html>
- [FIPS PUB 113]** NIST. *FIPS 113: Computer Data Authentication*. May 30, 1985. URL: <http://csrc.nist.gov/publications/fips/index.html>
- [FIPS PUB 180-2]** NIST. *FIPS 180-2: Secure Hash Standard*. August 1, 2002. URL: <http://csrc.nist.gov/publications/fips/index.html>
- [FORTEZZA CIPG]** NSA, Workstation Security Products. *FORTEZZA Cryptologic Interface Programmers Guide, Revision 1.52*. November 1985
- [GCS-API]** X/Open Company Ltd. Generic Cryptographic Service API (GCS-API), Base – Draft 2. February 14, 1995.
- [ISO/IEC 7816-1]** ISO/IEC 7816-1:2011. *Identification Cards – Integrated circuit cards -- Part 1: Cards with contacts -- Physical Characteristics*. 2011. URL: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=54089](http://www.iso.org/iso/catalogue_detail.htm?csnumber=54089).
- [ISO/IEC 7816-4]** ISO/IEC 7816-4:2013. *Identification Cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*. 2013. URL: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=54550](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54550).
- [ISO/IEC 8824-1]** ISO/IEC 8824-1:2008. *Abstract Syntax Notation One (ASN.1): Specification of Base Notation*. 2002. URL: [http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=54012](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54012)
- [ISO/IEC 8825-1]** ISO/IEC 8825-1:2008. Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). 2008. URL: [http://www.iso.org/iso/home/store/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54011&ics1=35&ics2=100&ics3=60)
- [ISO/IEC 9594-1]** ISO/IEC 9594-1:2008. *Information Technology – Open System Interconnection – The Directory: Overview of Concepts, Models and Services*. 2008. URL: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=53364](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53364)
- [ISO/IEC 9594-8]** ISO/IEC 9594-8:2008. *Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks*. 2008. URL: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=53372](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53372)
- [ISO/IEC 9796-2]** ISO/IEC 9796-2:2010. Information Technology – Security Techniques – Digital Signature Scheme Giving Message Recovery – Part 2: Integer factorization based mechanisms. 2010. URL: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=54788](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788)
- [Java MIDP]** Java Community Process. *Mobile Information Device Profile for Java 2 Micro Edition*. November 2002. URL: <http://jcp.org/jsr/detail/118.jsp>
- [MeT-PTD]** MeT. *MeT PTD Definition – Personal Trusted Device Definition, Version 1.0*. February 2003. URL: <http://www.mobiletransaction.org>
- [PCMCIA]** Personal Computer Memory Card International Association. *PC Card Standard, Release 2.1*. July 1993.
- [PKCS #1]** RSA Laboratories. *RSA Cryptography Standard, v2.1*. June 14, 2002. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>
- [PKCS #3]** RSA Laboratories. *Diffie-Hellman Key-Agreement Standard, v1.4*. November 1993.
- [PKCS #5]** RSA Laboratories. *Password-Based Encryption Standard, v2.0*. March 26, 1999. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf>
- [PKCS #7]** RSA Laboratories. *Cryptographic Message Syntax Standard, v1.6*. November 1997. URL: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-7/pkcs-7v16.pdf>

|     |                      |   |
|-----|----------------------|---|
| 136 | <b>[PKCS #8]</b>     | RSA Laboratories. <i>Private-Key Information Syntax Standard</i> , v1.2. November 1993. URL : <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn</a>   |
| 137 |                      |   |
| 138 | <b>[PKCS #11-UG]</b> | <i>PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40</i> . Edited by John Leiseboer and Robert Griffin. Latest 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-v2.40.html</a> .   |
| 139 |                      |   |
| 140 |                      |   |
| 141 | <b>[PKCS #12]</b>    | RSA Laboratories. <i>Personal Information Exchange Syntax Standard</i> , v1.0. June 1999. URL: <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf</a>  |
| 142 |                      |   |
| 143 | <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://www.rfc-editor.org/rfc/rfc1321.txt">http://www.rfc-editor.org/rfc/rfc1321.txt</a>  |
| 144 |                      |   |
| 145 |                      |   |
| 146 | <b>[RFC 3369]</b>    | R. Houseley. <i>RFC 3369: Cryptographic Message Syntax (CMS)</i> . August 2002. URL: <a href="http://www.rfc-editor.org/rfc/rfc3369.txt">http://www.rfc-editor.org/rfc/rfc3369.txt</a>  |
| 147 |                      |   |
| 148 | <b>[RFC 6149]</b>    | S. Turner and L. Chen. <i>RFC 6149: MD2 to Historic Status</i> . March, 2011. URL: <a href="http://www.rfc-editor.org/rfc/rfc6149.txt">http://www.rfc-editor.org/rfc/rfc6149.txt</a>  |
| 149 |                      |   |
| 150 | <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.  |
| 151 |                      |   |
| 152 |                      |   |
| 153 | <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.   |
| 154 |                      |   |
| 155 |                      |   |
| 156 | <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>  |
| 157 |                      |   |
| 158 | <b>[WIM]</b>         | WAP. <i>Wireless Identity Module</i> . – WAP-260-WIP-20010712.a. July 2001. URL: <a href="http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf">http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-260-wim-20010712-a.pdf</a>  |
| 159 |                      |   |
| 160 |                      |   |
| 161 | <b>[WPKI]</b>        | WAP. <i>Wireless Application Protocol: Public Key Infrastructure Definition</i> . – WAP-217-WPKI-20010424-a. April 2001. URL: <a href="http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf">http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-217-wpki-20010424-a.pdf</a> |
| 162 |                      |   |
| 163 |                      |   |
| 164 |                      |   |
| 165 | <b>[WTLS]</b>        | WAP. <i>Wireless Transport Layer Security Version</i> – WAP-261-WTLS-20010406-a. April 2001. URL: <a href="http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf">http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-261-wtls-20010406-a.pdf</a>                             |
| 166 |                      |   |
| 167 |                      |   |
| 168 |                      |   |
| 169 | <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)   |
| 170 |                      |   |
| 171 |                      |   |
| 172 | <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)  |
| 173 |                      |   |
| 174 |                      |   |
| 175 | <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)   |
| 176 |                      |   |
| 177 | <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)   |
| 178 |                      |   |
| 179 |                      |   |
| 180 |                      |   |

## 2 Mechanisms

### 2.1 PKCS #11 Mechanisms

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

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

Table 1, Mechanisms vs. Functions

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

| Mechanism                 | Functions               |                     |                            |        |                             |                     |        |
|---------------------------|-------------------------|---------------------|----------------------------|--------|-----------------------------|---------------------|--------|
|                           | Encrypt<br>&<br>Decrypt | Sign<br>&<br>Verify | SR<br>&<br>VR <sup>1</sup> | Digest | Gen.<br>Key/<br>Key<br>Pair | Wrap<br>&<br>Unwrap | Derive |
| CKM_CAST3_KEY_GEN         |                         |                     |                            |        | X                           |                     |        |
| CKM_CAST3_ECB             | X                       |                     |                            |        |                             | X                   |        |
| CKM_CAST3_CBC             | X                       |                     |                            |        |                             | X                   |        |
| CKM_CAST3_CBC_PAD         | X                       |                     |                            |        |                             | X                   |        |
| CKM_CAST3_MAC_GENERAL     |                         | X                   |                            |        |                             |                     |        |
| CKM_CAST3_MAC             |                         | X                   |                            |        |                             |                     |        |
| CKM_CAST128_KEY_GEN       |                         |                     |                            |        | X                           |                     |        |
| CKM_CAST128_ECB           | X                       |                     |                            |        |                             | X                   |        |
| CKM_CAST128_CBC           | X                       |                     |                            |        |                             | X                   |        |
| CKM_CAST128_CBC_PAD       | X                       |                     |                            |        |                             | X                   |        |
| CKM_CAST128_MAC_GENERAL   |                         | X                   |                            |        |                             |                     |        |
| CKM_CAST128_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                       |                     |                            |        |                             |                     |        |
| CKM_BATON_WRAP            |                         |                     |                            |        |                             | X                   |        |
| CKM_JUNIPER_KEY_GEN       |                         |                     |                            |        | X                           |                     |        |
| CKM_JUNIPER_ECB128        | X                       |                     |                            |        |                             |                     |        |
| CKM_JUNIPER_CBC128        | X                       |                     |                            |        |                             |                     |        |

| Mechanism                   | Functions         |               |                      |        |                   |               |        |
|-----------------------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|                             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| 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     |                   |               |                      |        | X                 |               |        |
| CKM_PBE_SHA1_CAST128_CBC    |                   |               |                      |        | X                 |               |        |
| CKM_PBE_SHA1_RC4_128        |                   |               |                      |        | X                 |               |        |
| CKM_PBE_SHA1_RC4_40         |                   |               |                      |        | X                 |               |        |
| CKM_PBE_SHA1_RC2_128_CBC    |                   |               |                      |        | X                 |               |        |
| CKM_PBE_SHA1_RC2_40_CBC     |                   |               |                      |        | X                 |               |        |
| CKM_PBA_SHA1_WITH_SHA1_HMAC |                   |               |                      |        | X                 |               |        |
| CKM_KEY_WRAP_SET_OAEP       |                   |               |                      |        |                   | X             |        |
| CKM_KEY_WRAP_LYNKS          |                   |               |                      |        |                   | X             |        |

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

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

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

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

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.

## 2.2 FORTEZZA timestamp

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

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.

## 2.3 KEA

### 2.3.1 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

### 2.3.2 KEA mechanism parameters

#### 2.3.2.1 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:

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

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

*pRandomA*      pointer to Ra data

*pRandomB*      pointer to Rb data

*ulPublicDataLen*      other party’s KEA public key size



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

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

### 243 2.3.3 KEA public key objects

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

247 *Table 3, KEA Public Key Object Attributes*

| Attribute                   | Data type   | Meaning   |
|-----------------------------|-------------|---|
| CKA_PRIME <sup>1,3</sup>    | Big integer | Prime $p$ (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 $y$                                  |

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

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

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

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

269

### 270 2.3.4 KEA private key objects

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

274 *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 11 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)}
};

```

### 2.3.5 KEA key pair generation

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

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.

## 2.3.6 KEA key derivation

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

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<br><i>isSender</i> | Value of big integer<br><i>pRandomB</i> | Token Action<br>(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\*:

---

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

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

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

## 2.4 RC2

### 2.4.1 Definitions

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 MAY take on values in the range 1-1024, inclusive. The effective number of bits in the RC2 search space is sometimes specified by an RC2 “version number”; this “version number” is *not* the same thing as the “effective number of bits”, however. There is a canonical way to convert from one to the other.

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

Mechanisms:

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

### 2.4.2 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                      |
|------------------------------|------------|------------------------------|
| 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 11 for footnotes

2.11 to match the policy used by other key derivation mechanisms such as **CKM\_SSL3\_MASTER\_KEY\_DERIVE**.

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

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

## 398 2.4.3 RC2 mechanism parameters

### 399 2.4.3.1 CK\_RC2\_PARAMS; CK\_RC2\_PARAMS\_PTR

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

```
402 typedef CK_ULONG CK_RC2_PARAMS;
```

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

### 404 2.4.3.2 CK\_RC2\_CBC\_PARAMS; CK\_RC2\_CBC\_PARAMS\_PTR

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

```
407 typedef struct CK_RC2_CBC_PARAMS {
408     CK_ULONG ulEffectiveBits;
409     CK_BYTE iv[8];
410 } CK_RC2_CBC_PARAMS;
```

411 The fields of the structure have the following meanings:

|     |                        |  |
|-----|------------------------|--|
| 412 | <i>ulEffectiveBits</i> | the effective number of bits in the RC2 search space     |
| 413 | <i>iv</i>              | the initialization vector (IV) for cipher block chaining |
| 414 |                        | mode   |

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

### 416 2.4.3.3 CK\_RC2\_MAC\_GENERAL\_PARAMS; 417 CK\_RC2\_MAC\_GENERAL\_PARAMS\_PTR

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

```
420 typedef struct CK_RC2_MAC_GENERAL_PARAMS {
421     CK_ULONG ulEffectiveBits;
422     CK_ULONG ulMacLength;
423 } CK_RC2_MAC_GENERAL_PARAMS;
```

424 The fields of the structure have the following meanings:

|     |                        |  |
|-----|------------------------|--|
| 425 | <i>ulEffectiveBits</i> | the effective number of bits in the RC2 search space |
| 426 | <i>ulMacLength</i>     | length of the MAC produced, in bytes                 |

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

#### 2.4.4 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.

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

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA\_KEY\_TYPE** attribute 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.

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

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA\_KEY\_TYPE** attribute 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.

## 2.4.7 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 MAY wrap and unwrap RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see **[PKCS #11-Curr]**, **Miscellaneous simple key derivation mechanisms** 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.

## 2.4.8 General-length RC2-MAC

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

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

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

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.

## 2.4.9 RC2-MAC

RC2-MAC, denoted by **CKM\_RC2\_MAC**, is a special case of the general-length RC2-MAC mechanism (see Section 2.4.8). 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 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.

## 2.5 RC4

### 2.5.1 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

### 2.5.2 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 11 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)}
};
```

### 2.5.3 RC4 key generation

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

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.



## 2.5.4 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.

## 2.6 RC5

### 2.6.1 Definitions

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

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

### 2.6.2 RC5 secret key objects

RC5 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_RC5**) hold RC5 keys. The following table defines the RC5 secret key object attributes, in addition to the common attributes defined 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 11 for footnotes

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

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

```

592 CK_ATTRIBUTE template[] = {
593     {CKA_CLASS, &class, sizeof(class)},
594     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
595     {CKA_TOKEN, &true, sizeof(true)},
596     {CKA_LABEL, label, sizeof(label)-1},
597     {CKA_ENCRYPT, &true, sizeof(true)},
598     {CKA_VALUE, value, sizeof(value)}
599 };

```

## 600 2.6.3 RC5 mechanism parameters

### 601 2.6.3.1 CK\_RC5\_PARAMS; CK\_RC5\_PARAMS\_PTR

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

```

604 typedef struct CK_RC5_PARAMS {
605     CK_ULONG ulWordsize;
606     CK_ULONG ulRounds;
607 } CK_RC5_PARAMS;

```

608 The fields of the structure have the following meanings:

|     |                   |                                      |
|-----|-------------------|--------------------------------------|
| 609 | <i>ulWordsize</i> | wordsize of RC5 cipher in bytes      |
| 610 | <i>ulRounds</i>   | number of rounds of RC5 encipherment |

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

### 612 2.6.3.2 CK\_RC5\_CBC\_PARAMS; CK\_RC5\_CBC\_PARAMS\_PTR

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

```

615 typedef struct CK_RC5_CBC_PARAMS {
616     CK_ULONG ulWordsize;
617     CK_ULONG ulRounds;
618     CK_BYTE_PTR pIv;
619     CK_ULONG ulIvLen;
620 } CK_RC5_CBC_PARAMS;

```

621 The fields of the structure have the following meanings:

|     |                   |  |
|-----|-------------------|--|
| 622 | <i>ulwordSize</i> | wordsize of RC5 cipher in bytes                          |
| 623 | <i>ulRounds</i>   | number of rounds of RC5 encipherment                     |
| 624 | <i>pIV</i>        | pointer to initialization vector (IV) for CBC encryption |
| 625 | <i>ulIVLen</i>    | length of initialization vector (must be same as         |
| 626 |                   | blocksize)   |

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

### 628 2.6.3.3 CK\_RC5\_MAC\_GENERAL\_PARAMS; 629 CK\_RC5\_MAC\_GENERAL\_PARAMS\_PTR

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

```

typedef struct CK_RC5_MAC_GENERAL_PARAMS {
    CK_ULONG ulWordsize;
    CK_ULONG ulRounds;
    CK_ULONG ulMacLength;
} CK_RC5_MAC_GENERAL_PARAMS;

```

The fields of the structure have the following meanings:

*ulwordSize*      wordsize of RC5 cipher in bytes

*ulRounds*      number of rounds of RC5 encipherment

*ulMacLength*      length of the MAC produced, in bytes

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

## 2.6.4 RC5 key generation

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

It does not have a parameter.

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

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

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

## 2.6.5 RC5-ECB

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

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

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

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA\_KEY\_TYPE** attributes 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 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<br>blocksize | Same as input length  | No final<br>part |
| C_WrapKey   | RC5 | Any                      | Input length rounded up to multiple of<br>blocksize           |                  |
| C_UnwrapKey | RC5 | Multiple of<br>blocksize | Determined by type of key being unwrapped<br>or CKA_VALUE_LEN |                  |

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

## 2.6.6 RC5-CBC

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

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

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

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

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

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

## 2.6.7 RC5-CBC with PKCS padding

RC5-CBC with PKCS padding, denoted **CKM\_RC5\_CBC\_PAD**, is a mechanism for single- and 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 an unwrap secret keys, this mechanism MAY wrap and unwrap RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys. The entries in the table below for data length constraints when wrapping and unwrapping keys 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.

## 2.6.8 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.

## 2.6.9 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 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.

## 2.7 General block cipher

### 2.7.1 Definitions

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

This section defines the key types "CKK\_DES", "CKK\_CAST", "CKK\_CAST3", "CKK\_CAST128", "CKK\_IDEA" and "CKK\_CDMF" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key 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

766 CKM\_CAST3\_MAC\_GENERAL  
 767 CKM\_CAST3\_CBC\_PAD  
 768 CKM\_CAST128\_KEY\_GEN  
 769 CKM\_CAST128\_ECB  
 770 CKM\_CAST128\_CBC  
 771 CKM\_CAST128\_MAC  
 772 CKM\_CAST128\_MAC\_GENERAL  
 773 CKM\_CAST128\_CBC\_PAD  
 774 CKM\_IDEA\_KEY\_GEN  
 775 CKM\_IDEA\_ECB  
 776 CKM\_IDEA\_MAC  
 777 CKM\_IDEA\_MAC\_GENERAL  
 778 CKM\_IDEA\_CBC\_PAD

## 779 2.7.2 DES secret key objects

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

783 *Table 20, DES Secret Key Object*

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

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

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

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

```
788 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
789 CK_KEY_TYPE keyType = CKK_DES;
790 CK_UTF8CHAR label[] = "A DES secret key object";
791 CK_BYTE value[8] = {...};
792 CK_BBOOL true = CK_TRUE;
793 CK_ATTRIBUTE template[] = {
794     {CKA_CLASS, &class, sizeof(class)},
795     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
796     {CKA_TOKEN, &true, sizeof(true)},
797     {CKA_LABEL, label, sizeof(label)-1},
798     {CKA_ENCRYPT, &true, sizeof(true)},
799     {CKA_VALUE, value, sizeof(value)}
800 };
```

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

## 804 2.7.3 CAST secret key objects

805 CAST secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_CAST**) hold CAST keys.  
 806 The following table defines the CAST secret key object attributes, in addition to the common attributes  
 807 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 11 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)}
};
```

## 2.7.4 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 11 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)}
};
```

## 2.7.5 CAST128 secret key objects

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



Table 23, CAST128 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 11 for footnotes

The following is a sample template for creating a CAST128 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)}
};
```

## 2.7.6 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 (16 bytes long) |

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

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

```
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)}
};
```

## 2.7.7 CDMF secret key objects

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

Table 25, CDMF Secret Key Object

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

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

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

## 2.7.8 General block cipher mechanism parameters

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

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

```
typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
```

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

## 2.7.9 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 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 be used. The CAST, CAST3, and CAST128 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.

## 2.7.10 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 MAY wrap and unwrap any secret key. Of course, a particular token MAY not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with 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 be used. The CAST, CAST3, and CAST128 ciphers have variable key sizes, and so for these ciphers, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these fields are not used.

## 2.7.11 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<br>blocksize | Same as input length  | No final<br>part |
| C_WrapKey   | <NAME> | Any                      | Input length rounded up to multiple of<br>blocksize           |                  |
| C_UnwrapKey | <NAME> | Any                      | Determined by type of key being<br>unwrapped or CKA_VALUE_LEN |                  |

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

## 2.7.12 General block cipher CBC with 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 MAY wrap and unwrap RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys. The entries in the table below for data length constraints when wrapping and unwrapping keys 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<br>blocksize | Between 1 and blocksize bytes shorter than input<br>length |
| C_WrapKey   | <NAME>   | Any                      | Input length rounded up to multiple of blocksize           |
| C_UnwrapKey | <NAME>   | Multiple of<br>blocksize | Between 1 and blocksize bytes shorter than input<br>length |

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

## 2.7.13 General-length general block cipher MAC

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

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

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

## 2.8 SKIPJACK

### 2.8.1 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\_OF64

CKM\_SKIPJACK\_CFB64

1030 CKM\_SKIPJACK\_CFB32  
1031 CKM\_SKIPJACK\_CFB16  
1032 CKM\_SKIPJACK\_CFB8  
1033 CKM\_SKIPJACK\_WRAP  
1034 CKM\_SKIPJACK\_PRIVATE\_WRAP  
1035 CKM\_SKIPJACK\_RELAYX

1036 **2.8.2 SKIPJACK secret key objects**

1037 SKIPJACK secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_SKIPJACK**) holds a  
1038 single-length MEK or a TEK. The following table defines the SKIPJACK secret object attributes, in  
1039 addition to the common attributes defined for this object class:

1040 *Table 31, SKIPJACK Secret Key Object*

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

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

1042

1043 SKIPJACK keys have 16 checksum bits, and these bits must be properly set. Attempting to create or  
1044 unwrap a SKIPJACK key with incorrect checksum bits MUST return an error.

1045 It is not clear that any tokens exist (or ever will exist) which permit an application to create a SKIPJACK  
1046 key with a specified value. Nonetheless, we provide templates for doing so.

1047 The following is a sample template for creating a SKIPJACK MEK secret key object:

```
1048 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1049 CK_KEY_TYPE keyType = CKK_SKIPJACK;  
1050 CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";  
1051 CK_BYTE value[12] = {...};  
1052 CK_BBOOL true = CK_TRUE;  
1053 CK_ATTRIBUTE template[] = {  
1054     {CKA_CLASS, &class, sizeof(class)},  
1055     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1056     {CKA_TOKEN, &true, sizeof(true)},  
1057     {CKA_LABEL, label, sizeof(label)-1},  
1058     {CKA_ENCRYPT, &true, sizeof(true)},  
1059     {CKA_VALUE, value, sizeof(value)}  
1060 };
```

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

```
1062 CK_OBJECT_CLASS class = CKO_SECRET_KEY;  
1063 CK_KEY_TYPE keyType = CKK_SKIPJACK;  
1064 CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";  
1065 CK_BYTE value[12] = {...};  
1066 CK_BBOOL true = CK_TRUE;  
1067 CK_ATTRIBUTE template[] = {  
1068     {CKA_CLASS, &class, sizeof(class)},  
1069     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1070     {CKA_TOKEN, &true, sizeof(true)},  
1071     {CKA_LABEL, label, sizeof(label)-1},  
1072     {CKA_ENCRYPT, &true, sizeof(true)},  
1073     {CKA_WRAP, &true, sizeof(true)},  
1074     {CKA_VALUE, value, sizeof(value)}  
1075 };
```



## 2.8.3 SKIPJACK Mechanism parameters

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

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

```
typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {  
    CK_ULONG ulPasswordLen;  
    CK_BYTE_PTR pPassword;  
    CK_ULONG ulPublicDataLen;  
    CK_BYTE_PTR pPublicData;  
    CK_ULONG ulPandGLen;  
    CK_ULONG ulQLen;  
    CK_ULONG ulRandomLen;  
    CK_BYTE_PTR pRandomA;  
    CK_BYTE_PTR pPrimeP;  
    CK_BYTE_PTR pBaseG;  
    CK_BYTE_PTR pSubprimeQ;  
} CK_SKIPJACK_PRIVATE_WRAP_PARAMS;
```

The fields of the structure have the following meanings:

|                        |   |
|------------------------|---|
| <i>ulPasswordLen</i>   | length of the password  |
| <i>pPassword</i>       | pointer to the buffer which contains the user-supplied password |
| <i>ulPublicDataLen</i> | other party's key exchange public key size                      |
| <i>pPublicData</i>     | pointer to other party's key exchange public key value          |
| <i>ulPandGLen</i>      | length of prime and base values                                 |
| <i>ulQLen</i>          | length of subprime value  |
| <i>ulRandomLen</i>     | size of random Ra, in bytes                                     |
| <i>pPrimeP</i>         | pointer to Prime, p, value                                      |
| <i>pBaseG</i>          | pointer to Base, b, value                                       |
| <i>pSubprimeQ</i>      | pointer to Subprime, q, value                                   |

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

### 2.8.3.2 CK\_SKIPJACK\_RELAYX\_PARAMS; CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR

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

```
typedef struct CK_SKIPJACK_RELAYX_PARAMS {  
    CK_ULONG ulOldWrappedXLen;  
    CK_BYTE_PTR pOldWrappedX;
```

```

1115     CK_ULONG ulOldPasswordLen;
1116     CK_BYTE_PTR pOldPassword;
1117     CK_ULONG ulOldPublicDataLen;
1118     CK_BYTE_PTR pOldPublicData;
1119     CK_ULONG ulOldRandomLen;
1120     CK_BYTE_PTR pOldRandomA;
1121     CK_ULONG ulNewPasswordLen;
1122     CK_BYTE_PTR pNewPassword;
1123     CK_ULONG ulNewPublicDataLen;
1124     CK_BYTE_PTR pNewPublicData;
1125     CK_ULONG ulNewRandomLen;
1126     CK_BYTE_PTR pNewRandomA;
1127 } CK_SKIPJACK_RELAYX_PARAMS;

```

1128 The fields of the structure have the following meanings:

|      |                           |   |
|------|---------------------------|---|
| 1129 | <i>ulOldWrappedLen</i>    | length of old wrapped key in bytes                                  |
| 1130 | <i>pOldWrappedX</i>       | pointer to old wrapper key  |
| 1131 | <i>ulOldPasswordLen</i>   | length of the old password  |
| 1132 | <i>pOldPassword</i>       | pointer to the buffer which contains the old user-supplied password |
| 1133 |                           |   |
| 1134 | <i>ulOldPublicDataLen</i> | old key exchange public key size                                    |
| 1135 | <i>pOldPublicData</i>     | pointer to old key exchange public key value                        |
| 1136 | <i>ulOldRandomLen</i>     | size of old random Ra in bytes                                      |
| 1137 | <i>pOldRandomA</i>        | pointer to old Ra data  |
| 1138 | <i>ulNewPasswordLen</i>   | length of the new password  |
| 1139 | <i>pNewPassword</i>       | pointer to the buffer which contains the new user-supplied password |
| 1140 |                           |   |
| 1141 | <i>ulNewPublicDataLen</i> | new key exchange public key size                                    |
| 1142 | <i>pNewPublicData</i>     | pointer to new key exchange public key value                        |
| 1143 | <i>ulNewRandomLen</i>     | size of new random Ra in bytes                                      |
| 1144 | <i>pNewRandomA</i>        | pointer to new Ra data  |

1145 **CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR** is a pointer to a **CK\_SKIPJACK\_RELAYX\_PARAMS**.

## 1146 2.8.4 SKIPJACK key generation

1147 The SKIPJACK key generation mechanism, denoted **CKM\_SKIPJACK\_KEY\_GEN**, is a key generation  
1148 mechanism for SKIPJACK. The output of this mechanism is called a Message Encryption Key (MEK).

1149 It does not have a parameter.

1150 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
1151 key.



## 2.8.5 SKIPJACK-ECB64

SKIPJACK-ECB64, denoted **CKM\_SKIPJACK\_ECB64**, is a mechanism for single- and multiple-part encryption and decryption with SKIPJACK in 64-bit electronic codebook mode as defined in FIPS PUB 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 can't specify a particular IV when encrypting. It MAY, of course, specify a particular IV when decrypting.

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

Table 32, SKIPJACK-ECB64: Data and Length

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

## 2.8.6 SKIPJACK-CBC64

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

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

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

Table 33, SKIPJACK-CBC64: Data and Length

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

## 2.8.7 SKIPJACK-OFB64

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

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

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

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 |

## 2.8.8 SKIPJACK-CFB64

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

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

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

*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 |

## 2.8.9 SKIPJACK-CFB32

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

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

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

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

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

## 2.8.10 SKIPJACK-CFB16

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

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

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

*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 |

## 2.8.11 SKIPJACK-CFB8

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

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

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 |

## 2.8.12 SKIPJACK-WRAP

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

## 2.8.13 SKIPJACK-PRIVATE-WRAP

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

## 2.8.14 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 2.8.13). 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.

# 2.9 BATON

## 2.9.1 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
- CKM\_BATON\_CBC128
- CKM\_BATON\_COUNTER
- CKM\_BATON\_SHUFFLE
- CKM\_BATON\_WRAP

## 2.9.2 BATON secret key objects

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

1243 Table 39, BATON Secret Key Object

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

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

1245

1246 BATON keys have 160 checksum bits, and these bits must be properly set. Attempting to create or

1247 unwrap a BATON key with incorrect checksum bits MUST return an error.

1248 It is not clear that any tokens exist (or will ever exist) which permit an application to create a BATON key

1249 with a specified value. Nonetheless, we provide templates for doing so.

1250 The following is a sample template for creating a BATON MEK secret key object:

```
1251 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1252 CK_KEY_TYPE keyType = CKK_BATON;
1253 CK_UTF8CHAR label[] = "A BATON MEK secret key object";
1254 CK_BYTE value[40] = {...};
1255 CK_BBOOL true = CK_TRUE;
1256 CK_ATTRIBUTE template[] = {
1257     {CKA_CLASS, &class, sizeof(class)},
1258     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1259     {CKA_TOKEN, &true, sizeof(true)},
1260     {CKA_LABEL, label, sizeof(label)-1},
1261     {CKA_ENCRYPT, &true, sizeof(true)},
1262     {CKA_VALUE, value, sizeof(value)}
1263 };
```

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

```
1265 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
1266 CK_KEY_TYPE keyType = CKK_BATON;
1267 CK_UTF8CHAR label[] = "A BATON TEK secret key object";
1268 CK_BYTE value[40] = {...};
1269 CK_BBOOL true = CK_TRUE;
1270 CK_ATTRIBUTE template[] = {
1271     {CKA_CLASS, &class, sizeof(class)},
1272     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1273     {CKA_TOKEN, &true, sizeof(true)},
1274     {CKA_LABEL, label, sizeof(label)-1},
1275     {CKA_ENCRYPT, &true, sizeof(true)},
1276     {CKA_WRAP, &true, sizeof(true)},
1277     {CKA_VALUE, value, sizeof(value)}
1278 };
```

1279 **2.9.3 BATON key generation**

1280 The BATON key generation mechanism, denoted **CKM\_BATON\_KEY\_GEN**, is a key generation

1281 mechanism for BATON. The output of this mechanism is called a Message Encryption Key (MEK).

1282 It does not have a parameter.

1283 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new

1284 key.

1285 **2.9.4 BATON-ECB128**

1286 BATON-ECB128, denoted **CKM\_BATON\_ECB128**, is a mechanism for single- and multiple-part

1287 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.9.5 BATON-ECB96

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

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

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 |

## 2.9.6 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.9.7 BATON-COUNTER

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

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

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 |

## 2.9.8 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.9.9 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 MAY 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.

## 2.10 JUNIPER

### 2.10.1 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
- CKM\_JUNIPER\_WRAP

### 2.10.2 JUNIPER secret key objects

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

Table 45, JUNIPER Secret Key Object

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

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

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

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

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

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

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

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

### 2.10.3 JUNIPER key generation

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

It does not have a parameter.

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

### 2.10.4 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.10.5 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.10.6 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.10.7 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 MAY NOT specify a particular IV when encrypting. It MAY, 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 |

## 2.10.8 JUNIPER WRAP

The JUNIPER wrap and unwrap mechanism, denoted **CKM\_JUNIPER\_WRAP**, is a function used to wrap and unwrap an MEK. It MAY 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.

## 2.11 MD2

### 2.11.1 Definitions

Mechanisms:

CKM\_MD2

CKM\_MD2\_HMAC

CKM\_MD2\_HMAC\_GENERAL

CKM\_MD2\_KEY\_DERIVATION

### 2.11.2 MD2 digest

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

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            |

### 2.11.3 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 MUST 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

## 2.11.4 MD2-HMAC

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

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

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

If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key **MUST** 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 **MAY** both be specified to be either **CK\_TRUE** or **CK\_FALSE**. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to **CK\_FALSE**, then the derived key **MUST** as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to **CK\_TRUE**, then the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_FALSE**, then the derived key **MUST**, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_TRUE**, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

## 2.12 MD5

### 2.12.1 Definitions

Mechanisms:

CKM\_MD5

CKM\_MD5\_HMAC

CKM\_MD5\_HMAC\_GENERAL

CKM\_MD5\_KEY\_DERIVATION

## 2.12.2 MD5 Digest

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

It does not have a parameter.

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

Table 52, MD5: Data Length

| Function | Data length | Digest length |
|----------|-------------|---------------|
| C_Digest | Any         | 16            |

## 2.12.3 General-length MD5-HMAC

The general-length MD5-HMAC mechanism, denoted **CKM\_MD5\_HMAC\_GENERAL**, is a mechanism for signatures and verification. It uses the HMAC construction, based on the MD5 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 MD5 is 16 bytes). Signatures (MACs) produced by this mechanism MUST be taken from the start of the full 16-byte HMAC output.

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 |

## 2.12.4 MD5-HMAC

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

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

## 2.12.5 MD5 key derivation

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

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

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

If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key MUST 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 MAY both be specified to either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key MUST as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key MUST, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

## 2.13 FASTHASH

### 2.13.1 Definitions

Mechanisms:

CKM\_FASTHASH

### 2.13.2 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            |

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

### 2.14.1 Definitions

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

Mechanisms:

CKM\_PBE\_MD2\_DES\_CBC  
CKM\_PBE\_MD5\_DES\_CBC  
CKM\_PBE\_MD5\_CAST\_CBC  
CKM\_PBE\_MD5\_CAST3\_CBC  
CKM\_PBE\_MD5\_CAST128\_CBC  
CKM\_PBE\_SHA1\_CAST128\_CBC  
CKM\_PBE\_SHA1\_RC4\_128  
CKM\_PBE\_SHA1\_RC4\_40  
CKM\_PBE\_SHA1\_RC2\_128\_CBC  
CKM\_PBE\_SHA1\_RC2\_40\_CBC

## 2.14.2 Password-based encryption/authentication mechanism parameters

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

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

```
typedef struct CK_PBE_PARAMS {  
    CK_BYTE_PTR pInitVector;  
    CK_UTF8CHAR_PTR pPassword;  
    CK_ULONG ulPasswordLen;  
    CK_BYTE_PTR pSalt;  
    CK_ULONG ulSaltLen;  
    CK_ULONG ulIteration;  
} CK_PBE_PARAMS;
```

The fields of the structure have the following meanings:

*pInitVector* pointer to the location that receives the 8-byte initialization vector (IV), if an IV is required

*pPassword* points to the password to be used in the PBE key generation

*ulPasswordLen* length in bytes of the password information

*pSalt* points to the salt to be used in the PBE key generation

*ulSaltLen* length in bytes of the salt information

*ulIteration* number of iterations required for the generation

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

### 2.14.3 MD2-PBE for DES-CBC

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

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

### 2.14.4 MD5-PBE for DES-CBC

MD5-PBE for DES-CBC, denoted **CKM\_PBE\_MD5\_DES\_CBC**, is a mechanism used for generating a DES secret key and an IV from a password and a salt value by using the MD5 digest algorithm and an 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 receives the 8-byte IV generated by the mechanism.



## 2.14.5 MD5-PBE for CAST-CBC

MD5-PBE for CAST-CBC, denoted **CKM\_PBE\_MD5\_CAST\_CBC**, is a mechanism used for generating a 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 receives 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.

## 2.14.6 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 receives 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.

## 2.14.7 MD5-PBE for CAST128-CBC

MD5-PBE for CAST128-CBC, denoted **CKM\_PBE\_MD5\_CAST128\_CBC**, is a mechanism used for generating a CAST128 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 receives the 8-byte IV generated by the mechanism

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

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

SHA-1-PBE for CAST128-CBC, denoted **CKM\_PBE\_SHA1\_CAST128\_**, is a mechanism used for generating a CAST128 secret key and an IV from a password and salt value using the SHA-1 digest algorithm and an iteration count. This functionality is analogous to that defined in PKCS #5 PBKDF1 for 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 receives the 8-byte IV generated by the mechanism

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

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

### 2.15.1 Definitions

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



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

Let  $H$  be a hash function built around a compression function  $f: \mathbb{Z}_2^u \times \mathbb{Z}_2^v \rightarrow \mathbb{Z}_2^u$  (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  $c$ th hash of  $D || I$ . That is, compute the hash of  $D || I$ ; compute the hash of that hash; etc.; continue in this fashion until a total of  $c$  hashes have been computed, each on the result of the previous hash.
  - b. Concatenate copies of  $A_i$  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.

## 2.15.2 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 receives an IV; for this mechanism, the contents of this field are ignored, since RC4 does not require an IV.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 2.16 RIPE-MD

### 2.16.1 Definitions

Mechanisms:

**CKM\_RIPEMD128**

**CKM\_RIPEMD128\_HMAC**

**CKM\_RIPEMD128\_HMAC\_GENERAL**

**CKM\_RIPEMD160**

**CKM\_RIPEMD160\_HMAC**

**CKM\_RIPEMD160\_HMAC\_GENERAL**

## 2.16.2 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            |

## 2.16.3 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 MUST 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 |

## 2.16.4 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 2.16.3.

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

## 2.16.5 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            |

## 2.16.6 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.

1759 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
1760 output. This length should be in the range 0-20 (the output size of RIPE-MD 160 is 20 bytes). Signatures  
1761 (MACs) produced by this mechanism MUST be taken from the start of the full 20-byte HMAC output.

1762 *Table 58, General-length RIPE-MD 160-HMAC: Data and Length*

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

1763 **2.16.7 RIPE-MD 160-HMAC**

1764 The RIPE-MD 160-HMAC mechanism, denoted **CKM\_RIPEMD160\_HMAC**, is a special case of the  
1765 general-length RIPE-MD 160HMAC mechanism in Section 2.16.6.

1766 It has no parameter, and produces an output of length 20.

1767 **2.17 SET**

1768 **2.17.1 Definitions**

1769 Mechanisms:

1770 **CKM\_KEY\_WRAP\_SET\_OAEP**

1771 **2.17.2 SET mechanism parameters**

1772 **2.17.2.1 CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS;**  
1773 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR**

1774 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** is a structure that provides the parameters to the  
1775 **CKM\_KEY\_WRAP\_SET\_OAEP** mechanism. It is defined as follows:

```
1776 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {  
1777     CK_BYTE bBC;  
1778     CK_BYTE_PTR pX;  
1779     CK_ULONG ulXLen;  
1780 } CK_KEY_WRAP_SET_OAEP_PARAMS;
```

1781 The fields of the structure have the following meanings:

1782 **bBC** block contents byte

1783 **pX** concatenation of hash of plaintext data (if present) and  
1784 extra data (if present)

1785 **ulXLen** length in bytes of concatenation of hash of plaintext data  
1786 (if present) and extra data (if present). 0 if neither is  
1787 present.

1788 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS\_PTR** is a pointer to a  
1789 **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS**.

1790 **2.17.3 OAEP key wrapping for SET**

1791 The OAEP key wrapping for SET mechanism, denoted **CKM\_KEY\_WRAP\_SET\_OAEP**, is a mechanism  
1792 for wrapping and unwrapping a DES key with an RSA key. The hash of some plaintext data and/or some

extra data MAY be wrapped together with the DES key. This mechanism is defined in the SET protocol 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 [PKCS #11-Curr], **Miscellaneous simple key derivation mechanisms**. Note that if the inputs to **C\_UnwrapKey** are such that the extra data is not returned (e.g. the buffer supplied in the **CK\_KEY\_WRAP\_SET\_OAEP\_PARAMS** structure is **NULL\_PTR**), then the unwrapped key object **MUST** 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 **MUST** NOT be larger than 128 bytes), rather than calling **C\_UnwrapKey** twice. Each call of **C\_UnwrapKey** with **CKM\_KEY\_WRAP\_SET\_OAEP** requires an RSA decryption operation to be performed, and this computational overhead MAY be avoided by this means.

## 2.18 LYNKS

### 2.18.1 Definitions

Mechanisms:

**CKM\_KEY\_WRAP\_LYNKS**

### 2.18.2 LYNKS key wrapping

The LYNKS key wrapping mechanism, denoted **CKM\_KEY\_WRAP\_LYNKS**, is a mechanism for wrapping and unwrapping secret keys with DES keys. It MAY wrap any 8-byte secret key, and it produces 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*<sub>1</sub> and *sum*<sub>2</sub>, to 0
2. Loop through the bytes of *K* from first to last.
3. Set *sum*<sub>1</sub> = *sum*<sub>1</sub> + the key byte (treat the key byte as a number in the range 0-255).
4. Set *sum*<sub>2</sub> = *sum*<sub>2</sub> + *sum*<sub>1</sub>.
5. Encrypt *K* with *W* in ECB mode, obtaining an encrypted key, *E*.
6. Concatenate the last 6 bytes of *E* with *sum*<sub>2</sub>, representing *sum*<sub>2</sub> most-significant bit first. The result is an 8-byte block, *T*
7. Encrypt *T* with *W* in ECB mode, obtaining an encrypted checksum, *C*.
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

1834

1835 An implementation is a conforming implementation if it meets the conditions specified in one or more  
1836 server profiles specified in **[PKCS #11-Prof]**.

1837 A PKCS #11 implementation SHALL be a conforming PKCS #11 implementation.

1838 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL  
1839 conform to all normative statements within the clauses specified for that profile and for any subclauses to  
1840 each of those clauses .

---

## 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  
Jeff Bartell, Semper Fortis Solutions LLC  
Peter Bartok, Venafi, Inc.  
Anthony Berglas, Cryptsoft  
Joseph Brand, Semper Fortis Solutions LLC  
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  
John Green, QuintessenceLabs  
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)  
Wang Jingman, Feitan Technologies  
Andrey Jivsov, Symantec Corp.  
Mark Joseph, P6R  
Stefan Kaesar, Infineon Technologies  
Greg Kazmierczak, Wave Systems Corp.  
Mark Knight, Thales e-Security  
Darren Krahn, Google Inc.



1882 Alex Krasnov, Infineon Technologies AG  
1883 Dina Kurktchi-Nimeh, Oracle  
1884 Mark Lambiase, SecureAuth Corporation  
1885 Lawrence Lee, GoTrust Technology Inc.  
1886 John Leiseboer, QuintessenceLabs  
1887 Sean Leon, Infineon Technologies  
1888 Geoffrey Li, Infineon Technologies  
1889 Howie Liu, Infineon Technologies  
1890 Hal Lockhart, Oracle  
1891 Robert Lockhart, Thales e-Security  
1892 Dale Moberg, Axway Software  
1893 Darren Moffat, Oracle  
1894 Valery Osheter, SafeNet, Inc.  
1895 Sean Parkinson, EMC  
1896 Rob Philpott, EMC  
1897 Mark Powers, Oracle  
1898 Ajai Puri, SafeNet, Inc.  
1899 Robert Relyea, Red Hat  
1900 Saikat Saha, Oracle  
1901 Subhash Sankuratipati, NetApp  
1902 Anthony Scarpino, Oracle  
1903 Johann Schoetz, Infineon Technologies AG  
1904 Rayees Shamsuddin, Wave Systems Corp.  
1905 Radhika Siravara, Oracle  
1906 Brian Smith, Mozilla Corporation  
1907 David Smith, Venafi, Inc.  
1908 Ryan Smith, Futurex  
1909 Jerry Smith, US Department of Defense (DoD)  
1910 Oscar So, Oracle  
1911 Graham Steel, Cryptosense  
1912 Michael Stevens, QuintessenceLabs  
1913 Michael StJohns, Individual  
1914 Jim Susoy, P6R  
1915 Sander Temme, Thales e-Security  
1916 Kiran Thota, VMware, Inc.  
1917 Walter-John Turnes, Gemini Security Solutions, Inc.  
1918 Stef Walter, Red Hat  
1919 James Wang, Vormetric  
1920 Jeff Webb, Dell  
1921 Peng Yu, Feitian Technologies  
1922 Magda Zdunkiewicz, Cryptsoft  
1923 Chris Zimman, Individual

---

## Appendix B. Manifest constants

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

- include/pkcs11-v3.0/pkcs11.h
- include/pkcs11-v3.0/pkcs11t.h
- include/pkcs11-v3.0/pkcs11f.h

These files are linked from the "[Additional artifacts](#)" section at the top of this specification.

---

## Appendix C. Revision History

| Revision | Date          | Editor      | Changes Made   |
|----------|---------------|-------------|--|
| wd01     | 20 April 2019 | Dieter Bong | <ul style="list-style-type: none"><li>- All CAST5 definitions removed</li><li>- Replaced reference to [PKCS11-Base] table 10 by [PKCS11-Base] table 11 throughout whole document</li></ul> |
| wd02     | May 28, 2019  | Tony Cox    | Final cleanup of front introductory texts and links prior to CSPRD   |
|          |               |             |  |