PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification
Version 2.40 Plus Errata 01

OASIS Standard Incorporating Draft 01 of Errata 01

09 December 2015

Specification URIs
This version:
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/csd01/pkcs11-hist-v2.40-errata01-csd01-complete.doc (Authoritative)
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/csd01/pkcs11-hist-v2.40-errata01-csd01-complete.html
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/csd01/pkcs11-hist-v2.40-errata01-csd01-complete.pdf

Previous version:
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/os/pkcs11-hist-v2.40-os.doc (Authoritative)
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/os/pkcs11-hist-v2.40-os.html

Latest version:
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/pkcs11-hist-v2.40-errata01-complete.doc (Authoritative)
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/pkcs11-hist-v2.40-errata01-complete.html
http://docs.oasis-open.org/pkcs11/pkcs11-hist/v2.40/errata01/pkcs11-hist-v2.40-errata01-complete.pdf

Technical Committee:
OASIS PKCS 11 TC

Chairs:
Valerie Fenwick (valerie.fenwick@oracle.com), Oracle
Robert Relyea (rrelyea@redhat.com), Red Hat

Editors:
Susan Gleeson (susan.gleeson@oracle.com), Oracle
Chris Zimman (chris@wmpp.com), Individual
Robert Griffin (robert.griffin@emc.com), EMC Corporation
Tim Hudson (tjh@cryptsoft.com), Cryptsoft Pty Ltd

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

Related work:

This specification replaces or supersedes:


This specification is related to:

- Normative computer language definition files for PKCS #11 v2.40:
  - http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/csd01/include/pkcs11-v2.40/pkcs11.h
  - http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/csd01/include/pkcs11-v2.40/pkcs11t.h
  - http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/errata01/csd01/include/pkcs11-v2.40/pkcs11f.h

Abstract:

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

Status:

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

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

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

Citation format:

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

[PKCS11-Hist-v2.40]
Notices

Copyright © OASIS Open 2015. 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 ................................................................................................................................. 9
   1.1 Description of this Document ................................................................................................. 9
   1.2 Terminology .............................................................................................................................. 9
   1.3 Definitions ............................................................................................................................... 9
   1.4 Normative References ........................................................................................................... 10
   1.5 Non-Normative References .................................................................................................... 10
2  Mechanisms .................................................................................................................................. 13
   2.1 PKCS #11 Mechanisms .......................................................................................................... 13
   2.2 FORTEZZA timestamp ......................................................................................................... 16
   2.3 KEA ........................................................................................................................................ 16
      2.3.1 Definitions ......................................................................................................................... 16
      2.3.2 KEA mechanism parameters ............................................................................................. 16
         2.3.2.1 CK KEA_DERIVE_PARAMS; CK KEA_DERIVE_PARAMS_PTR ........................................ 16
      2.3.3 KEA public key objects .................................................................................................. 17
      2.3.4 KEA private key objects .................................................................................................. 18
      2.3.5 KEA key pair generation .................................................................................................. 18
      2.3.6 KEA key derivation .......................................................................................................... 19
   2.4 RC2 ....................................................................................................................................... 20
      2.4.1 Definitions ......................................................................................................................... 20
      2.4.2 RC2 secret key objects .................................................................................................... 20
      2.4.3 RC2 mechanism parameters ............................................................................................. 21
         2.4.3.1 CK RC2_PARAMS; CK RC2_PARAMS_PTR ...................................................................... 21
         2.4.3.2 CK RC2_CBC_PARAMS; CK RC2 CBC_PARAMS_PTR ........................................................ 21
         2.4.3.3 CK RC2 MAC GENERAL_PARAMS; CK RC2 MAC GENERAL_PARAMS_PTR ..................... 21
      2.4.4 RC2 key generation ........................................................................................................ 22
      2.4.5 RC2 ECB .......................................................................................................................... 22
      2.4.6 RC2 CBC .......................................................................................................................... 23
      2.4.7 RC2 CBC with PKCS padding ......................................................................................... 23
      2.4.8 General-length RC2 MAC ............................................................................................... 24
      2.4.9 RC2 MAC ........................................................................................................................ 24
   2.5 RC4 ....................................................................................................................................... 25
      2.5.1 Definitions ......................................................................................................................... 25
      2.5.2 RC4 secret key objects ..................................................................................................... 25
      2.5.3 RC4 key generation .......................................................................................................... 25
      2.5.4 RC4 mechanism ............................................................................................................... 26
   2.6 RC5 ....................................................................................................................................... 26
      2.6.1 Definitions ......................................................................................................................... 26
      2.6.2 RC5 secret key objects ..................................................................................................... 26
      2.6.3 RC5 mechanism parameters ............................................................................................. 27
         2.6.3.1 CK RC5_PARAMS; CK RC5_PARAMS_PTR ...................................................................... 27
         2.6.3.2 CK RC5_CBC_PARAMS; CK RC5_CBC_PARAMS_PTR ........................................................ 27
         2.6.3.3 CK RC5_MAC GENERAL_PARAMS; CK RC5 MAC GENERAL_PARAMS_PTR ..................... 28
      2.6.4 RC5 key generation ........................................................................................................ 28
      2.6.5 RC5 ECB .......................................................................................................................... 28
2.9 BATON........................................................................................................... 44
  2.9.1 Definitions................................................................................................. 44
  2.9.2 BATON secret key objects ................................................................. 45
  2.9.3 BATON key generation .......................................................... 46
  2.9.4 BATON-ECB128 ................................................................................. 46
  2.9.5 BATON-ECB96 ................................................................................. 46
  2.9.6 BATON-CBC128 .............................................................................. 46
  2.9.7 BATON-COUNTER ........................................................................... 47
  2.9.8 BATON-SHUFFLE .............................................................................. 47

2.8 SKIPJACK...................................................................................................... 39
  2.8.1 Definitions................................................................................................. 39
  2.8.2 SKIPJACK secret key objects .............................................................. 39
  2.8.3 SKIPJACK Mechanism parameters ................................................. 40
    2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS; CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR...................................... 40
    2.8.3.2 CK_SKIPJACK_RELAYX_PARAMS; CK_SKIPJACK_RELAYX_PARAMS_PTR.............................................. 41
  2.8.4 SKIPJACK key generation ............................................................ 42
  2.8.5 SKIPJACK-ECB64 ............................................................................ 42
  2.8.6 SKIPJACK-CBC64 ............................................................................ 42
  2.8.7 SKIPJACK-OFB64 ............................................................................ 43
  2.8.8 SKIPJACK-CFB64 ............................................................................ 43
  2.8.9 SKIPJACK-CFB32 ............................................................................ 43
  2.8.10 SKIPJACK-CFB16 .......................................................................... 43
  2.8.11 SKIPJACK-CFB8 ........................................................................... 44
  2.8.12 SKIPJACK-WRAP ........................................................................... 44
  2.8.13 SKIPJACK-PRIVATE-WRAP ............................................................ 44
  2.8.14 SKIPJACK-RELAYX .......................................................... 44

2.7 General block cipher ................................................................................ 31
  2.7.1 Definitions................................................................................................. 31
  2.7.2 DES secret key objects ............................................................................. 32
  2.7.3 CAST secret key objects ........................................................................... 33
  2.7.4 CAST3 secret key objects ....................................................................... 33
  2.7.5 CAST128 (CAST5) secret key objects .................................................. 34
  2.7.6 IDEA secret key objects ......................................................................... 34
  2.7.7 CDMF secret key objects ....................................................................... 35
  2.7.8 General block cipher mechanism parameters ..................................... 35
    2.7.8.1 CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR.................................................. 35
  2.7.9 General block cipher key generation ..................................................... 35
  2.7.10 General block cipher ECB ................................................................. 36
  2.7.11 General block cipher CBC ................................................................... 37
  2.7.12 General block cipher CBC with PKCS padding ................................... 37
  2.7.13 General-length general block cipher MAC ......................................... 38
  2.7.14 General block cipher MAC ................................................................. 38

2.6 General........................................................................................................ 29
  2.6.6 RC5-CBC ............................................................................................... 29
  2.6.7 RC5-CBC with PKCS padding .............................................................. 30
  2.6.8 General-length RC5-MAC ..................................................................... 30
  2.6.9 RC5-MAC ............................................................................................... 31
1 Introduction

1.1 Description of this Document

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

All text is normative unless otherwise labeled.

1.2 Terminology

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

1.3 Definitions

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

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATON</td>
<td>MISSI’s BATON block cipher.</td>
</tr>
<tr>
<td>CAST</td>
<td>Entrust Technologies’ proprietary symmetric block cipher</td>
</tr>
<tr>
<td>CAST3</td>
<td>Entrust Technologies’ proprietary symmetric block cipher</td>
</tr>
<tr>
<td>CAST5</td>
<td>Another name for Entrust Technologies’ symmetric block cipher</td>
</tr>
<tr>
<td>CAST128</td>
<td>Entrust Technologies’ symmetric block cipher.</td>
</tr>
<tr>
<td>CDMF</td>
<td>Commercial Data Masking Facility, a block encipherment method specified by International Business Machines Corporation and based on DES.</td>
</tr>
<tr>
<td>CMS</td>
<td>Cryptographic Message Syntax (see RFC 3369)</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard, as defined in FIPS PUB 46-3</td>
</tr>
<tr>
<td>ECB</td>
<td>Electronic Codebook mode, as defined in FIPS PUB 81.</td>
</tr>
<tr>
<td>FASTHASH</td>
<td>MISSI’s FASTHASH message-digesting algorithm.</td>
</tr>
<tr>
<td>IDEA</td>
<td>Ascom Systec’s symmetric block cipher.</td>
</tr>
<tr>
<td>IV</td>
<td>Initialization Vector.</td>
</tr>
<tr>
<td>JUNIPER</td>
<td>MISSI’s JUNIPER block cipher.</td>
</tr>
<tr>
<td>KEA</td>
<td>MISSI’s Key Exchange Algorithm.</td>
</tr>
<tr>
<td>LYNKS</td>
<td>A smart card manufactured by SPYRUS.</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>MD2</td>
<td>RSA Security’s MD2 message-digest algorithm, as defined in RFC 6149.</td>
</tr>
<tr>
<td>MD5</td>
<td>RSA Security’s MD5 message-digest algorithm, as defined in RFC 1321.</td>
</tr>
</tbody>
</table>
PRF  Pseudo random function.
RSA  The RSA public-key cryptosystem.
RC2  RSA Security’s RC2 symmetric block cipher.
RC4  RSA Security’s proprietary RC4 symmetric stream cipher.
RC5  RSA Security’s RC5 symmetric block cipher.
SET  The Secure Electronic Transaction protocol.
SHA-1 The (revised) Secure Hash Algorithm with a 160-bit message digest, as defined in FIPS PUB 180-2.
SKIPJACK MISSI’s SKIPJACK block cipher.

1.4 Normative References


1.5 Non-Normative References


Standards Track Work Product Copyright © OASIS Open 2015. All Rights Reserved.
## 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*

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Encrypt &amp; Decrypt</th>
<th>Sign &amp; Verify</th>
<th>SR &amp; VR</th>
<th>Digest</th>
<th>Gen. Key/Key Pair</th>
<th>Wrap &amp; Unwrap</th>
<th>Derive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_FORTEZZA_TIMESTAMP</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM KEA_KEY_PAIR_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM KEA_KEY_DERIVE</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC2_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC2 ECB</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC2_CBC</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC2_CBC_PAD</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC2_MAC_GENERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC2_MAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC4 KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC4</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC5_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC5 ECB</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC5_CBC</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC5_CBC_PAD</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC5_MAC_GENERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM RC5_MAC</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM DES KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM DES ECB</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM DES CBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM DES_CBC_PAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM DES MAC GENERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM DES_MAC</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM CAST KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM CAST ECB</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM CAST_CBC</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM CAST_CBC_PAD</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Mechanism</th>
<th>Encrypt &amp; Decrypt</th>
<th>Sign &amp; Verify</th>
<th>SR &amp; VR</th>
<th>Digest</th>
<th>Gen. Key/Key Pair</th>
<th>Wrap &amp; Unwrap</th>
<th>Derive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_CAST_MACGENERAL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST_MAC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST3_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CAST3_ECB</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CAST3_CBC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CAST3_CBC_PAD</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CAST3_MAC_GENERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST3_MAC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST128_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(CKM_CAST5_KEY_GEN)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST128_ECB</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(CKM_CAST5_ECB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST128_CBC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(CKM_CAST5_CBC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST128_CBC_PAD</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(CKM_CAST5_CBC_PAD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CAST128_MAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(CKM_CAST5_MAC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_IDEA_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_IDEA_ECB</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_IDEA_CBC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_IDEA_CBC_PAD</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_IDEA_MAC_GENERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_IDEA_MAC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_CDMF_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CDMF_ECB</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CDMF_CBC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CDMF_CBC_PAD</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CDMF_MAC_GENERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_CDMF_MAC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_ECB64</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_CBC64</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_OFB64</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_CFB64</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_CFB32</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_CFB16</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_CFB8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK.WRAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_PRIVATE_WRAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_SKIPJACK_RELAYX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X*</td>
<td></td>
</tr>
<tr>
<td>CKM_BATON_KEY_GEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CKM_BATON_ECB128</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_BATON_ECB96</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mechanism</td>
<td>Encrypt &amp; Decrypt</td>
<td>Sign &amp; Verify</td>
<td>SR &amp; VR</td>
<td>Digest</td>
<td>Gen. Key/Key Pair</td>
<td>Wrap &amp; Unwrap</td>
<td>Derive</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>------------------</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>CKM_BATON_CBC128</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_BATON_COUNTER</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_BATON_SHUFFLE</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_BATON_WRAP</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_JUNIPER_KEY_GEN</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_JUNIPER_ECB128</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_JUNIPER_CBC128</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_JUNIPER_COUNTER</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_JUNIPER_SHUFFLE</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_JUNIPER_WRAP</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_MD2</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_MD2_HMAC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_MD2_KEY_DERIVATION</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_MD5</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_MD5_HMAC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_MD5_KEY_DERIVATION</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RIPEMD128</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RIPEMD128_HMAC_GENERAL</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RIPEMD128_HMAC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RIPEMD160</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RIPEMD160_HMAC_GENERAL</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RIPEMD160_HMAC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_FASTHASH</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_MD2_DES_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_MD5_DES_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_MD5_CAST_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_MD5_CAST3_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_MD5_CAST128_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_SHA1_CAST128_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_SHA1_CAST5_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_SHA1_RC4_128</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_SHA1_RC4_40</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_SHA1_RC2_128_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBE_SHA1_RC2_40_CBC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_PBA_SHA1_WITH_SHA1_HMAC</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_KEY_WRAP_SET_OAEP</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_KEY_WRAP_LYNKS</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 SR = SignRecover, VR = VerifyRecover.
2 Single-part operations only.
3 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 \texttt{ulMinKeyLen} and \texttt{ulMaxKeyLen} fields of the \texttt{CK_MECHANISM_INFO} structure, then those fields have no meaning for that particular mechanism.

### 2.2 FORTEZZA timestamp

The FORTEZZA timestamp mechanism, denoted \texttt{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*

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input Length</th>
<th>Output Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign(^1)</td>
<td>DSA private key</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>C_Verify(^1)</td>
<td>DSA public key</td>
<td>20,40(^2)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

\(^1\) Single part operations only
\(^2\) Data length, signature length

For this mechanism, the \texttt{ulMinKeySize} and \texttt{ulMaxKeySize} fields of the \texttt{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 \texttt{CK\_KEY\_TYPE} as used in the \texttt{CKA\_KEY\_TYPE} attribute of key objects.

Mechanisms:

\begin{verbatim}
    CKM\_KEA\_KEY\_PAIR\_GEN
    CKM\_KEA\_KEY\_DERIVE
\end{verbatim}

#### 2.3.2 KEA mechanism parameters

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

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

```c
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

pPublicData pointer to other party’s KEA public key value

CK_KEA_DERIVE_PARAMS_PTR is a pointer to a CK_KEA_DERIVE_PARAMS.

2.3.3 KEA public key objects

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

Table 3, KEA Public Key Object Attributes

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

\(^{1}\) Refer to [PKCS #11-Base] table 10 for footnotes

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

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

```c
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK KEA;
CK_UTF8CHAR label[] = “A KEA public key object”;
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
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_PRIME, prime, sizeof(prime)},
    {CKA_SUBPRIME, subprime, sizeof(subprime)},
    {CKA_BASE, base, sizeof(base)},
    {CKA_VALUE, value, sizeof(value)}
};
```
### 2.3.4 KEA private key objects

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_PRIME</td>
<td>Big integer</td>
<td>Prime $p$ (512 to 1024 bits, in steps of 64 bits)</td>
</tr>
<tr>
<td>CKA_SUBPRIME</td>
<td>Big integer</td>
<td>Subprime $q$ (160 bits)</td>
</tr>
<tr>
<td>CKA_BASE</td>
<td>Big integer</td>
<td>Base $g$ (512 to 1024 bits, in steps of 64 bits)</td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>Big integer</td>
<td>Private value $x$</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 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:

```c
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_KEYA;
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)},
  {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_KEYA_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_DEAKEA_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_KEY_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_KEY_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>
<thead>
<tr>
<th>Value of boolean isSender</th>
<th>Value of big integer pRandomB</th>
<th>Token Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_TRUE</td>
<td>0</td>
<td>Compute KEA R_a value, store it in pRandomA, return CKR_OK. No derived key object is created.</td>
</tr>
<tr>
<td>CK_TRUE</td>
<td>1</td>
<td>Compute KEA R_a value, store it in pRandomA, derive key value using e-mail mode, create key object, return CKR_OK.</td>
</tr>
<tr>
<td>CK_TRUE</td>
<td>&gt;1</td>
<td>Compute KEA R_a value, store it in pRandomA, derive key value using full mode, create key object, return CKR_OK.</td>
</tr>
<tr>
<td>CK_FALSE</td>
<td>0</td>
<td>Compute KEA R_b value, store it in pRandomB, return CKR_OK. No derived key object is created.</td>
</tr>
<tr>
<td>CK_FALSE</td>
<td>1</td>
<td>Derive key value using e-mail mode, create key object, return CKR_OK.</td>
</tr>
<tr>
<td>CK_FALSE</td>
<td>&gt;1</td>
<td>Derive key value using full mode, create key object, return CKR_OK.</td>
</tr>
</tbody>
</table>

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

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

- The `CKA_SENSITIVE` and `CKA_EXTRACTABLE` attributes in the template for the new key 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 keysizes 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

Note that the rules regarding the `CKA_SENSITIVE`, `CKA_EXTRACTABLE`, `CKA_ALWAYS_SENSITIVE`, and `CKA_NEVER_EXTRACTABLE` attributes have changed in version 2.11 to match the policy used by other key derivation mechanisms such as `CKM_SSL3_MASTER_KEY_DERIVE`. 
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE(^1,4,6,7)</td>
<td>Byte array</td>
<td>Key value (1 to 128 bytes)</td>
</tr>
<tr>
<td>CKA_VALUE_LEN(^2,3)</td>
<td>CK_ULONG</td>
<td>Length in bytes of key value</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 for footnotes.

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

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

2.4.3 RC2 mechanism parameters

2.4.3.1 CK_RC2_PARAMS; CK_RC2_PARAMS_PTR

CK_RC2_PARAMS provides the parameters to the CKM_RC2_ECB and CMK_RC2_MAC mechanisms. It holds the effective number of bits in the RC2 search space. It is defined as follows:

```c
typedef CK_ULONG CK_RC2_PARAMS;
```

CK_RC2_PARAMS_PTR is a pointer to a CK_RC2_PARAMS.

2.4.3.2 CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR

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

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

The fields of the structure have the following meanings:

- `ulEffectiveBits`: the effective number of bits in the RC2 search space
- `iv`: the initialization vector (IV) for cipher block chaining mode

CK_RC2_CBC_PARAMS_PTR is a pointer to a CK_RC2_CBC_PARAMS.

2.4.3.3 CK_RC2_MACGENERAL_PARAMS;

CK_RC2_MACGENERAL_PARAMS_PTR

CK_RC2_MACGENERAL_PARAMS is a structure that provides the parameters to the CKM_RC2_MAC mechanism. It is defined as follows:

```c
typedef struct CK_RC2_MAC_GENERAL_PARAMS {
    CK_ULONG ulEffectiveBits;
};
```
The fields of the structure have the following meanings:

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

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

### 2.4.4 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**

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>RC2</td>
<td>Any</td>
<td>Input length rounded up to multiple of 8</td>
<td></td>
</tr>
</tbody>
</table>

---

pkcs11-hist-v2.40-errata01-csd01-complete
Standards Track Work Product Copyright © OASIS Open 2015. All Rights Reserved. 09 December 2015
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 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_VALUE 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>RC2</td>
<td>Any</td>
<td>Input length rounded up to multiple of 8</td>
<td></td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Determined by type of key being unwrapped or CKA_VALUE_LEN</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>RC2</td>
<td>Any</td>
<td>Input length rounded up to multiple of 8</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Between 1 and 8 bytes shorter than input length</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>RC2</td>
<td>Any</td>
<td>Input length rounded up to multiple of 8</td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>RC2</td>
<td>Multiple of 8</td>
<td>Between 1 and 8 bytes shorter than input length</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>RC2</td>
<td>Any</td>
<td>0-8, as specified in parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>RC2</td>
<td>Any</td>
<td>0-8, as specified in parameters</td>
</tr>
</tbody>
</table>

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-MA 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>RC2</td>
<td>Any</td>
<td>4</td>
</tr>
</tbody>
</table>
For this mechanism, the \texttt{ulMinKeySize} and \texttt{ulMaxKeySize} fields of the \texttt{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

- \texttt{CKM\_RC4\_KEY\_GEN}
- \texttt{CKM\_RC4}

#### 2.5.2 RC4 secret key objects

RC4 secret key objects (object class \texttt{CKO\_SECRET\_KEY}, key type \texttt{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>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE\textsuperscript{1,4,6,7}</td>
<td>Byte array</td>
<td>Key value (1 to 256 bytes)</td>
</tr>
<tr>
<td>CKA_VALUE_LEN\textsuperscript{2,5,6}</td>
<td>CK_ULONG</td>
<td>Length in bytes of key value</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 for footnotes

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

```c
CK\_OBJECT\_CLASS class = CKO\_SECRET\_KEY;
CK\_KEY\_TYPE keyType = CKK\_RC4;
CK\_UTF8\_CHAR label[] = "An RC4 secret key object";
CK\_BYTE value[] = {...};
CK\_B\_BOOL true - CK\_TRUE;
CK\_ATTRIBUTE template[] = {
    {CKA\_CLASS, &class, sizeof(class)},
    {CKA\_KEY\_TYPE, &keyType, sizeof(keyType)},
    {CKA\_LABEL, label, sizeof(label)-1},
    {CKA\_ENC\_CRYPT, &true, sizeof(true)},
    {CKA\_VALUE, value, sizeof(value)}
};
```

#### 2.5.3 RC4 key generation

The RC4 key generation mechanism, denoted \texttt{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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>RC4</td>
<td>Any</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>RC4</td>
<td>Any</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE⁴.⁶.⁷</td>
<td>Byte array</td>
<td>Key value (0 to 255 bytes)</td>
</tr>
<tr>
<td>CKA_VALUE_LEN².³.⁶</td>
<td>CK_ULONG</td>
<td>Length in bytes of key value</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 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;
CK_ATTRIBUTE template[] = {
    CKA_CLASS, &class, sizeof(class)),
    CKA_KEY_TYPE, &keyType, sizeof(keyType)),
    CKA_TOKEN, &true, sizeof(true)),
    CKA_LABEL, label, sizeof(label)-1),
    CKA_ENCRYPT, &true, sizeof(true)),
    CKA_VALUE, value, sizeof(value))
};

2.6.3 RC5 mechanism parameters

2.6.3.1 CK_RC5_PARAMS; CK_RC5_PARAMS_PTR

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

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

The fields of the structure have the following meanings:

ulWordsize wordsize of RC5 cipher in bytes
ulRounds number of rounds of RC5 encipherment

CK_RC5_PARAMS_PTR is a pointer to a CK_RC5_PARAMS.

2.6.3.2 CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR

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

typedef struct CK_RC5_CBC_PARAMS {
    CK_ULONG ulWordsize;
    CK_ULONG ulRounds;
    CK_BYTE_PTR pIV;
    CK_ULONG ulIVLen;
} CK_RC5_CBC_PARAMS;

The fields of the structure have the following meanings:

ulWordSize wordsize of RC5 cipher in bytes
ulRounds number of rounds of RC5 encipherment
pIV pointer to initialization vector (IV) for CBC encryption
ulIVLen length of initialization vector (must be same as blocksize)

CK_RC5_CBC_PARAMS_PTR is a pointer to a CK_RC5_CBC_PARAMS.
2.6.3.3 CK_RC5_MAC_GENERAL_PARAMS;

CK_RC5_MAC_GENERAL_PARAMS_PTR

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

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

The fields of the structure have the following meanings:

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
ciphering 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
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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>RC5</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>RC5</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>RC5</td>
<td>Any</td>
<td>Input length rounded up to multiple of blocksize</td>
<td></td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>RC5</td>
<td>Multiple of blocksize</td>
<td>Determined by type of key being unwrapped or CKA_VALUE_LEN</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>RC5</td>
<td>Any</td>
<td>Input length rounded up to multiple of blocksize</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>RC5</td>
<td>Multiple of blocksize</td>
<td>Between 1 and blocksize bytes shorter than input length</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>RC5</td>
<td>Any</td>
<td>Input length rounded up to multiple of blocksize</td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>RC5</td>
<td>Multiple of blocksize</td>
<td>Between 1 and blocksize bytes shorter than input length</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>RC5</td>
<td>Any</td>
<td>0-blocksize, as specified in parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>RC5</td>
<td>Any</td>
<td>0-blocksize, as specified in parameters</td>
</tr>
</tbody>
</table>

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 CKM_RC5_MAC_GENERAL_PARAMS parameter, it takes a CKM_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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>RC5</td>
<td>Any</td>
<td>RC5 wordsize = [blocksize/2]</td>
</tr>
<tr>
<td>C_Verify</td>
<td>RC5</td>
<td>Any</td>
<td>RC5 wordsize = [blocksize/2]</td>
</tr>
</tbody>
</table>

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 (CAST5), IDEA and CDMF block ciphers are described together here. Each of these ciphers has the following mechanisms, which are described in a templatized form.

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

Mechanisms:

- CKM_DES_KEY_GEN
- CKM_DES_ECB
- CKM_DES_CBC
- CKM_DES_MAC
- CKM_DES_MAC_GENERAL
- CKM_DES_CBC_PAD
- CKM_CDMF_KEY_GEN
- CKM_CDMF_ECB
- CKM_CDMF_CBC
- CKM_CDMF_MAC
- CKM_CDMF_MAC_GENERAL
- CKM_CDMF_CBC_PAD
- CKM_DES_OFB64
- CKM_DES_OFB8
- CKM_DES_CFB64
- CKM_DES_CFB8
- CKM_CAST_KEY_GEN
- CKM_CAST_ECB
- CKM_CAST_CBC
- CKM_CAST_MAC
- CKM_CAST_MAC_GENERAL
The following is a sample template for creating a DES secret key object:

```c
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES;
CK_UTF8CHAR label[] = "A DES 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_VALUE, value, sizeof(value)},
}
```
{CKA_LABEL, label, sizeof(label)-1},
{CKA_ENCRYPT, &true, sizeof(true)},
{CKA_VALUE, value, sizeof(value)};

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

2.7.3 CAST secret key objects

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

Table 21, CAST Secret Key Object Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>Byte array</td>
<td>Key value (1 to 8 bytes)</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>CK_ULONG</td>
<td>Length in bytes of key value</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>Byte array</td>
<td>Key value (1 to 8 bytes)</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>CK_ULONG</td>
<td>Length in bytes of key value</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 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 (CAST5) secret key objects

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>1,4,6,7</td>
<td>Byte array, Key value (1 to 16 bytes)</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>2,3,6</td>
<td>CK ULONG, Length in bytes of key value</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 for footnotes

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

```c
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>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>1,4,6,7</td>
<td>Byte array, Key value (16 bytes long)</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 for footnotes

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

```c
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[] = {
```
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 defined for this object class:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>Byte array</td>
<td>Key value (8 bytes long)</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 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:

```c
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_MC_GENERAL_PARAMS; CK_MC_GENERAL_PARAMS_PTR

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

```c
typedef CK_ULONG CK_MC_GENERAL_PARAMS;
```

CK_MC_GENERAL_PARAMS_PTR is a pointer to a CK_MC_GENERAL_PARAMS.

2.7.9 General block cipher key generation

Cipher <NAME> has a key generation mechanism, "<NAME> key generation", denoted by

```c
CKM_<NAME>_KEY_GEN.
```
This mechanism does not have a parameter.

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

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

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

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

For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the CK_MECHANISM_INFO structure MAY be used. The CAST, CAST3, and CAST128 (CAST5) ciphers have variable key sizes, and so for the key generation mechanisms for these ciphers, the ulMinKeySize and ulMaxKeySize fields of the CK_MECHANISM_INFO structure specify the supported range of key sizes, in bytes. For the DES, DES3 (triple-DES), IDEA and CDMF ciphers, these fields and 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>&lt;NAME&gt;</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>&lt;NAME&gt;</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>Input length rounded up to multiple of blocksize</td>
<td></td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>Determined by type of key being unwrapped or CKA_VALUE_LEN</td>
<td></td>
</tr>
</tbody>
</table>

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

2.7.11 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>&lt;NAME&gt;</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>&lt;NAME&gt;</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>Input length rounded up to multiple of blocksize</td>
<td></td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>Determined by type of key being unwrapped or CKA_VALUE_LEN</td>
<td></td>
</tr>
</tbody>
</table>

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

2.7.12 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 to not apply to wrapping
and unwrapping private keys.

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>&lt;NAME&gt;</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>&lt;NAME&gt;</td>
<td>Multiple of blocksize</td>
<td>Same as input length</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>Input length rounded up to multiple of blocksize</td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>Determined by type of key being unwrapped or CKA_VALUE_LEN</td>
</tr>
</tbody>
</table>
C_Encrypt  <NAME>  Any  Input length rounded up to multiple of blocksize
C_Decrypt  <NAME>  Multiple of blocksize  Between 1 and blocksize bytes shorter than input length
C_WrapKey  <NAME>  Any  Input length rounded up to multiple of blocksize
C_UnwrapKey  <NAME>  Multiple of blocksize  Between 1 and blocksize bytes shorter than input length

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

### 2.7.13 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>0-blocksize, depending on parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>0-blocksize, depending on parameters</td>
</tr>
</tbody>
</table>

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

### 2.7.14 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>[blocksize/2]</td>
</tr>
<tr>
<td>C_Verify</td>
<td>&lt;NAME&gt;</td>
<td>Any</td>
<td>[blocksize/2]</td>
</tr>
</tbody>
</table>

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

### 2.8 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_OFB64**
- **CKM_SKIPJACK_CFB64**
- **CKM_SKIPJACK_CFB32**
- **CKM_SKIPJACK_CFB16**
- **CKM_SKIPJACK_CFB8**
- **CKM_SKIPJACK_WRAP**
- **CKM_SKIPJACK_PRIVATE_WRAP**
- **CKM_SKIPJACK_RELAYX**

#### 2.8.2 SKIPJACK secret key objects

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CKA_VALUE</strong></td>
<td>Byte array</td>
<td>Key value (12 bytes long)</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 for footnotes

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

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

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

```c
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SKIPJACK;
CK_UTF8CHAR label[] = "A SKIPJACK MEK secret key object";
CK_BYTE value[12] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
```
The following is a sample template for creating a SKIPJACK TEK secret key object:

```c
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SKIPJACK;
CK_UTF8CHAR label[] = "A SKIPJACK TEK secret key object";
CK_BYTE value[12] = {...};
CK_BBOOL true = CK_TRUE;

CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_WRAP, &true, sizeof(true)},
    {CKA_VALUE, value, sizeof(value)}
};
```

### 2.8.3 SKIPJACK Mechanism parameters

#### 2.8.3.1 CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

`CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR` 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:

- `ulPasswordLen` length of the password
- `pPassword` pointer to the buffer which contains the user-supplied password
- `ulPublicDataLen` other party’s key exchange public key size
- `pPublicData` pointer to other party’s key exchange public key value
- `ulPandGLen` length of prime and base values
- `ulQLen` length of subprime value
- `ulRandomLen` size of random Ra, in bytes
1113     \textit{pPrimeP} pointer to Prime, p, value
1114     \textit{pBaseG} pointer to Base, b, value
1115     \textit{pSubprimeQ} pointer to Subprime, q, value
1116
1117     \texttt{CK\_SKIPJACK\_PRIVATE\_WRAP\_PARAMS\_PTR} is a pointer to a
1118     \texttt{CK\_PRIVATE\_WRAP\_PARAMS}.
1119
1120     \texttt{2.8.3.2 CK\_SKIPJACK\_RELAYX\_PARAMS;}
1121     \texttt{CK\_SKIPJACK\_RELAYX\_PARAMS\_PTR}
1122
1123     \texttt{CK\_SKIPJACK\_RELAYX\_PARAMS} is a structure that provides the parameters to the
1124     \texttt{CKM\_SKIPJACK\_RELAYX} mechanism. It is defined as follows:
1125
1126     
1127     \begin{verbatim}
1128     typedef struct CK_SKIPJACK_RELAYX_PARAMS {
1129         CK_ULONG ulOldWrappedXLen;
1130         CK_BYTE_PTR pOldWrappedX;
1131         CK_ULONG ulOldPasswordLen;
1132         CK_BYTE_PTR pOldPassword;
1133         CK_ULONG ulOldPublicDataLen;
1134         CK_BYTE_PTR pOldPublicData;
1135         CK_ULONG ulOldRandomLen;
1136         CK_BYTE_PTR pOldRandomA;
1137         CK_ULONG ulNewPasswordLen;
1138         CK_BYTE_PTR pNewPassword;
1139         CK_ULONG ulNewPublicDataLen;
1140         CK_BYTE_PTR pNewPublicData;
1141         CK_ULONG ulNewRandomLen;
1142         CK_BYTE_PTR pNewRandomA;
1143     } CK_SKIPJACK_RELAYX_PARAMS;
1144     \end{verbatim}
1145
1146     The fields of the structure have the following meanings:
1147
1148     \textit{ulOldWrappedLen} length of old wrapped key in bytes
1149
1150     \textit{pOldWrappedX} pointer to old wrapper key
1151
1152     \textit{ulOldPasswordLen} length of the old password
1153
1154     \textit{pOldPassword} pointer to the buffer which contains the old user-supplied password
1155
1156     \textit{ulOldPublicDataLen} old key exchange public key size
1157
1158     \textit{pOldPublicData} pointer to old key exchange public key value
1159
1160     \textit{ulOldRandomLen} size of old random Ra in bytes
1161
1162     \textit{pOldRandomA} pointer to old Ra data
1163
1164     \textit{ulNewPasswordLen} length of the new password
1165
1166     \textit{pNewPassword} pointer to the buffer which contains the new user-supplied password
ulNewPublicKeyDataLen     new key exchange public key size
pNewPublicKeyData          pointer to new key exchange public key value
ulNewRandomLen             size of new random Ra in bytes
pNewRandomA                pointer to new Ra data

CK_SKIPJACK_RELAYX_PARAMS_PTR is a pointer to a CK_SKIPJACK_RELAYX_PARAMS.

2.8.4 SKIPJACK key generation

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

It does not have a parameter.

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

2.8.5 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 cant 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 8</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 4</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 4</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 4</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 4</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 4</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>SKIPJACK</td>
<td>Multiple of 4</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>Byte array</td>
<td>Key value (40 bytes long)</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 for footnotes

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

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

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

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

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

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

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

It does not have a parameter.

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

2.9.4 BATON-ECB128

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

It has a parameter, a 24-byte initialization vector. During an encryption operation, this IV is set to some value generated by the token – in other words, the application 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>BATON</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>BATON</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>BATON</td>
<td>Multiple of 12</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>BATON</td>
<td>Multiple of 12</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>BATON</td>
<td>Multiple of 12</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>BATON</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>BATON</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>BATON</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>BATON</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

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
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>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALUE</td>
<td>Byte array</td>
<td>Key value (40 bytes long)</td>
</tr>
</tbody>
</table>

Refer to [PKCS #11-Base] table 10 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:

```c
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_VALUE, value, sizeof(value)}
};
```

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

```c
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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>JUNIPER</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>JUNIPER</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>JUNIPER</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>JUNIPER</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Input length</th>
<th>Output length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Encrypt</td>
<td>JUNIPER</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td>JUNIPER</td>
<td>Multiple of 16</td>
<td>Same as input length</td>
<td>No final part</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Table 49, JUNIPER-SHUFFLE: Data and Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>C_Encrypt</td>
</tr>
<tr>
<td>C_Decrypt</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Table 50, MD2: Data Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>C_Digest</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-16, depending on parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-16, depending on parameters</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Data length</th>
<th>Digest length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Digest</td>
<td>Any</td>
<td>16</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-16, depending on parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-16, depending on parameters</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Input length</th>
<th>Digest length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Digest</td>
<td>Any</td>
<td>40</td>
</tr>
</tbody>
</table>

### 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_CAST5_CBC
CKM_PBE_MD5_CAST128_CBC
CKM_PBE_SHA1_CAST5_CBC
CKM_PBE_SHA1_CAST128_CBC
CKM_PBE_SHA1_RC4_128
CKM_PBE_SHA1_RC4_40
CKM_PBE_SHA1_RC2_1
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:

```c
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 (CAST5-CBC)

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

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

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

2.14.8 SHA-1-PBE for CAST128-CBC (CAST5-CBC)

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

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

The length of the CAST128 (CAST5) key generated by this mechanism MAY be specified in the supplied template; if it is not present in the template, it defaults to 8 bytes.
2.15 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 \( f : \mathbb{Z}_2^n \times \mathbb{Z}_2^* \rightarrow \mathbb{Z}_2^m \) (that is, H has a chaining variable and output of length \( m \) bits, and the message input to the compression function of H is \( n \) 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 \( v/8 \) 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 \( v/8 \) 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 \( l=S||P \) to be the concatenation of \( S \) and \( P \).

5. Set \( j=\lceil n/u \rceil \).

6. For \( i=1, 2, \ldots, j \), do the following:
   a. Set \( A_i=H(D||l) \), the \( c \)th hash of \( D||l \). That is, compute the hash of \( D||l \), 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 \( A \) to create a string \( B \) of length \( v \) bits (the final copy of \( A \) MAY be truncated to create \( B \)).
   c. Treating \( l \) as a concatenation \( l_0, l_1, \ldots, l_{k-1} \) of \( v \)-bit blocks, where \( k=\lceil v/8 \rceil \), \( \lceil p/8 \rceil \), modify \( l \) by setting \( l_j=(l_j+B+1) \mod 2^v \) for each \( j \). To perform this addition, treat each \( v \)-bit block as a binary number represented most-significant bit first.

7. Concatenate \( A_1, A_2, \ldots, A_j \) 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 \texttt{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 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 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**
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>
<thead>
<tr>
<th>Function</th>
<th>Data length</th>
<th>Digest length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Digest</td>
<td>Any</td>
<td>16</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-16, depending on parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-16, depending on parameters</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Function</th>
<th>Data length</th>
<th>Digest length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Digest</td>
<td>Any</td>
<td>20</td>
</tr>
</tbody>
</table>
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.

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Key type</th>
<th>Data length</th>
<th>Signature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Sign</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-20, depending on parameters</td>
</tr>
<tr>
<td>C_Verify</td>
<td>Generic secret</td>
<td>Any</td>
<td>0-20, depending on parameters</td>
</tr>
</tbody>
</table>

2.16.7 RIPE-MD 160-HMAC

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

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

2.17 SET

2.17.1 Definitions

Mechanisms:

CKM_KEY_WRAP_SET_OAEP

2.17.2 SET mechanism parameters

2.17.2.1 CK_KEY_WRAP_SET_OAEP_PARAMS;

CK_KEY_WRAP_SET_OAEP_PARAMS_PTR

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

```c
typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {
    CK_BYTE bBC;
    CK_BYTE_PTR pX;
    CK_ULONG ulXLen;
} CK_KEY_WRAP_SET_OAEP_PARAMS;
```

The fields of the structure have the following meanings:

- `bBC` block contents byte
- `pX` concatenation of hash of plaintext data (if present) and extra data (if present)
- `ulXLen` length in bytes of concatenation of hash of plaintext data (if present) and extra data (if present). 0 if neither is present.

CK_KEY_WRAP_SET_OAEP_PARAMS_PTR is a pointer to a CK_KEY_WRAP_SET_OAEP_PARAMS.
2.17.3 OAEP key wrapping for SET

The OAEP key wrapping for SET mechanism, denoted CKM_KEY_WRAP_SET_OAEP, is a mechanism 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, $\text{sum}_1$ and $\text{sum}_2$, to 0
2. Loop through the bytes of $K$ from first to last.
3. Set $\text{sum}_1 = \text{sum}_1 + \text{the key byte}$ (treat the key byte as a number in the range 0-255).
4. Set $\text{sum}_2 = \text{sum}_2 + \text{sum}_1$.
5. Encrypt $K$ with $W$ in ECB mode, obtaining an encrypted key, $E$.
6. Concatenate the last 6 bytes of $E$ with $\text{sum}_2$, representing $\text{sum}_2$ most-significant bit first. The result is an 8-byte block, $T$.
7. Encrypt $T$ with $W$ in ECB mode, obtaining an encrypted checksum, $C$.
8. Concatenate $E$ with the last 2 bytes of $C$ to obtain the wrapped key.

When unwrapping a key with this mechanism, if the cryptographic checksum does not check out properly, an error is returned. In addition, if a DES key or CDMF key is unwrapped with this mechanism, the parity bits on the wrapped key must be set appropriately. If they are not set properly, an error is returned.
3 PKCS #11 Implementation Conformance

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

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

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

The following individuals have participated in the creation of this specification and are gratefully 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.
1937  Magda Zdunkiewicz, Cryptsoft
1938  Chris Zimman, Individual
Appendix B. Manifest constants

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

```c
/*
 * Copyright OASIS Open 2014. All rights reserved.
 * OASIS trademark, IPR and other policies apply.
 * http://www.oasis-open.org/policies-guidelines/ipr
 */

#define CKK_KA 0x00000005
#define CKK_RC2 0x00000011
#define CKK_RC4 0x00000012
#define CKK_DES 0x00000013
#define CKK_CAST 0x00000016
#define CKK_CAST3 0x00000017
#define CKK_CAST5 0x00000018
#define CKK_CAST128 0x00000018
#define CKK_RC5 0x00000019
#define CKK_IDEA 0x0000001A
#define CKK_SKIPJACK 0x0000001B
#define CKK_BATON 0x0000001C
#define CKK_JUNIPER 0x0000001D
#define CKM_MD2_RSA_PKCS 0x00000004
#define CKM_MD5_RSA_PKCS 0x00000005
#define CKM_RIPEMD128_RSA_PKCS 0x00000007
#define CKM_RIPEMD160_RSA_PKCS 0x00000008
#define CKM_RC2_KEY_GEN 0x00000100
#define CKM_RC2_CBC 0x00000102
#define CKM_RC2_MAC 0x00000103
#define CKM_RC2_CBC_PAD 0x00000105
#define CKM_RC4_KEY_GEN 0x00000110
#define CKM_RC4 0x00000111
#define CKM_DES_KEY_GEN 0x00000120
#define CKM_DES_ECB 0x00000121
#define CKM_DES_CBC 0x00000122
#define CKM_DES_MAC 0x00000123
#define CKM_MD2_HMAC 0x00000201
#define CKM_MD2_HMAC_GENERAL 0x00000202
#define CKM_MD5_HMAC 0x00000210
#define CKM_MD5_HMAC_GENERAL 0x00000211
#define CKM_RIPEMD128_HMAC 0x00000231
#define CKM_RIPEMD128_HMAC_GENERAL 0x00000232
#define CKM_RIPEMD160_HMAC 0x00000240
#define CKM_RIPEMD160_HMAC_GENERAL 0x00000241
#define CKM_CAST_KEY_GEN 0x00000300
#define CKM_CAST_CBC 0x00000302
#define CKM_CAST_MAC 0x00000303
#define CKM_CAST_CBC_PAD 0x00000305
#define CKM_CAST3_KEY_GEN 0x00000310
```
#define CKM_CAST3_ECB 0x00000311
1998
#define CKM_CAST3_CBC 0x00000312
1999
#define CKM_CAST3_MAC 0x00000313
2000
#define CKM_CAST128_KEY_GEN 0x00000320
2001
#define CKM_CAST128_CBC 0x00000321
2002
#define CKM_CAST128_MAC 0x00000323
2003
#define CKM_CAST5_KEY_GEN 0x00000320
2004
#define CKM_CAST5_ECB 0x00000321
2005
#define CKM_CAST5_CBC 0x00000322
2006
#define CKM_CAST5_MAC 0x00000323
2007
#define CKM_CAST5_MAC_GENERAL 0x00000324
2008
#define CKM_CAST5_CBC_PAD 0x00000325
2009
#define CKM_CAST128_KEY_GEN 0x00000320
2010
#define CKM_CAST128_CBC 0x00000322
2011
#define CKM_CAST128_MAC 0x00000323
2012
#define CKM_CAST128_MAC_GENERAL 0x00000324
2013
#define CKM_CAST128_CBC_PAD 0x00000325
2014
#define CKM_RC5_KEY_GEN 0x00000330
2015
#define CKM_RC5_ECB 0x00000331
2016
#define CKM_RC5_CBC 0x00000332
2017
#define CKM_RC5_MAC 0x00000333
2018
#define CKM_RC5_MAC_GENERAL 0x00000334
2019
#define CKM_RC5_CBC_PAD 0x00000335
2020
#define CKM_IDEA_KEY_GEN 0x00000340
2021
#define CKM_IDEA_ECB 0x00000341
2022
#define CKM_IDEA_CBC 0x00000342
2023
#define CKM_IDEA_MAC 0x00000343
2024
#define CKM_IDEA_MAC_GENERAL 0x00000344
2025
#define CKM_IDEA_CBC_PAD 0x00000345
2026
#define CKM_MD5_KEY_DERIVATION 0x00000390
2027
#define CKM_MD2_KEY_DERIVATION 0x00000391
2028
#define CKM_PBE_MD2_DES_CBC 0x000003A0
2029
#define CKM_PBE_MD5_DES_CBC 0x000003A1
2030
#define CKM_PBE_MD5_CAST_CBC 0x000003A2
2031
#define CKM_PBE_MD5_CAST3_CBC 0x000003A3
2032
#define CKM_PBE_MD5_CAST5_CBC 0x000003A4
2033
#define CKM_PBE_MD5_CAST128_CBC 0x000003A5
2034
#define CKM_PBE_SHA1_CAST5_CBC 0x000003A6
2035
#define CKM_PBE_SHA1_CAST128_CBC 0x000003A7
2036
#define CKM_PBE_SHA1_CAST128_CBC 0x000003A8
2037
#define CKM_KEY_WRAP_LYNKS 0x00000400
2038
#define CKM_KEY_WRAP_SET_OAEP 0x00000401
2039
#define CKM_SKIPJACK_KEY_GEN 0x00000400
2040
#define CKM_SKIPJACK_ECB64 0x00000401
2041
#define CKM_SKIPJACK_CBC64 0x00000402
2042
#define CKM_SKIPJACK_CBC64 0x00000403
2043
#define CKM_SKIPJACK_OFB64 0x00000404
2044
#define CKM_SKIPJACK_CFB64 0x00000405
2045
#define CKM_SKIPJACK_CFB32 0x00000406
2046
#define CKM_SKIPJACK_CFB16 0x00000407
2047
#define CKM_SKIPJACK_CFB8 0x00000408
2048
#define CKM_SKIPJACK_WRAP 0x00000409
2049
#define CKM_SKIPJACK_PRIVATE_WRAP 0x0000040A
2050
#define CKM_SKIPJACK_RELAYX 0x0000040B
2051
#define CKM_KEA_KEY_PAIR_GEN 0x00000410
2052
#define CKM_KEA_KEY_DERIVE 0x00000411
2053
#define CKM_FORTEZZA_TIMESTAMP 0x00000412
2054
#define CKM_BATON_KEY_GEN 0x00000410
2055
#define CKM_BATON_ECB128 0x00000411
2056
#define CKM_BATON_ECB96 0x00000412
2057
#define CKM_BATON_CBC128 0x00000413
2058
#define CKM_BATON_COUNTER 0x00000414
2059
#define CKM_BATON_COUNTER 0x00000415
2060
#define CKM_BATON_COUNTER 0x00000416
2061
```c
#define CKM_BATON_SHUFFLE 0x00001035
#define CKM_BATON_WRAP 0x00001036
#define CKM_JUNIPER_KEY_GEN 0x00001060
#define CKM_JUNIPER_ECB128 0x00001061
#define CKM_JUNIPER_CBC128 0x00001062
#define CKM_JUNIPER_COUNTER 0x00001063
#define CKM_JUNIPER_SHUFFLE 0x00001064
#define CKM_JUNIPER_WRAP 0x00001065
#define CKM_FASTHASH 0x00001070
```

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

- include/pkcs11-v2.40/pkcs11.h
- include/pkcs11-v2.40/pkcs11t.h
- include/pkcs11-v2.40/pkcs11f.h

These files are linked from the Related Work section at the top of this specification.
# Appendix C. Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Editor</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>wd01</td>
<td>May 16, 2013</td>
<td>Susan Gleeson</td>
<td>Initial Template import</td>
</tr>
<tr>
<td>wd02</td>
<td>July 7, 2013</td>
<td>Susan Gleeson</td>
<td>Fix references, add participants list, minor cleanup</td>
</tr>
<tr>
<td>wd03</td>
<td>October 27, 2013</td>
<td>Robert Griffin</td>
<td>Final participant list and other editorial changes for Committee Specification Draft</td>
</tr>
<tr>
<td>csd01</td>
<td>October 30, 2013</td>
<td>OASIS</td>
<td>Committee Specification Draft</td>
</tr>
<tr>
<td>wd04</td>
<td>February 19, 2014</td>
<td>Susan Gleeson</td>
<td>Incorporate changes from v2.40 public review</td>
</tr>
<tr>
<td>wd05</td>
<td>February 20, 2014</td>
<td>Susan Gleeson</td>
<td>Regenerate table of contents (oversight from wd04)</td>
</tr>
<tr>
<td>WD06</td>
<td>February 21, 2014</td>
<td>Susan Gleeson</td>
<td>Remove CKM_PKCS5_PBKD2 from the mechanisms in Table 1.</td>
</tr>
<tr>
<td>csd02</td>
<td>April 23, 2014</td>
<td>OASIS</td>
<td>Committee Specification Draft</td>
</tr>
<tr>
<td>csd02a</td>
<td>Sep 3 2014</td>
<td>Robert Griffin</td>
<td>Updated revision history and participant list in preparation for Committee Specification ballot</td>
</tr>
<tr>
<td>wd07</td>
<td>Nov 3 2014</td>
<td>Robert Griffin</td>
<td>Editorial corrections</td>
</tr>
<tr>
<td>os</td>
<td>Apr 14 2015</td>
<td>OASIS</td>
<td>OASIS Standard</td>
</tr>
<tr>
<td>os-rev01</td>
<td>Dec 9 2015</td>
<td>Robert Griffin / Tim Hudson</td>
<td>Change bar edits corresponding to Errata01</td>
</tr>
</tbody>
</table>